

N 69 20192

Technical Report 69-85

January 1969

A Translator System  
for the  
EULER Programming Language

by  
Victor B. Schneider



**CASE FILE  
COPY**

**UNIVERSITY OF MARYLAND  
COMPUTER SCIENCE CENTER  
COLLEGE PARK, MARYLAND**



Technical Report 69-85

January 1969

A Translator System  
for the  
EULER Programming Language  
by  
Victor B. Schneider

The research for this paper was supported in part by N.A.S.A. grant  
NsG-398 to the Computer Science Center of the University of Maryland.

## Abstract

### A Translator System for the EULER Programming Language.

EULER is a generalization of the ALGOL programming language, combining the ALGOL capabilities with many features similar to McCarthy's LISP language. In this paper, a new syntax is presented for EULER, together with a notation that specifies how EULER is translated. A fast, pushdown-automaton model was used as a basis for designing the translator system, and the design methods used for a portion of the translator are presented. Syntactic methods of machine-independent optimization of the translated language are discussed, and a syntactic notation for designing "extendible compilers" in terms of the EULER language is introduced.

The second section of the report is a documentation of the intermediate language into which the EULER is translated. Algorithms for executing the commands of this intermediate language are described, and versions of the programmed algorithms, rewritten in a subset of EULER, are presented.

The appendices contain listings of the actual translator and intermediate language interpreter for the EULER system. These listings consist of FORTRAN IV-MAP programs written for the IBM 7094 computer.

## T A B L E O F C O N T E N T S

|   |    |
|---|----|
| Part I. The EULER Translator . . . . .                              | 1  |
| Introduction . . . . .  | 1  |
| An EULER Translation Grammar . . . . .                              | 2  |
| Syntactic Methods for Optimizing Expressions . . . . .              | 7  |
| Some Syntactic Methods for Extending the EULER Language . . . . .   | 8  |
| How A Section of the Translator Was Designed . . . . .              | 13 |
| Error Messages and Scans in the Translator . . . . .                | 15 |
| A Missing Feature of the Translator . . . . .                       | 16 |
| Translation Times and Memory Requirements . . . . .                 | 17 |
| Part II. The EULER Interpretive System . . . . .                    | 18 |
| Introduction . . . . .  | 18 |
| Structures Within the Interpreter . . . . .                         | 19 |
| Typing and Storage of Data . . . . .                                | 20 |
| Insertion of Data . . . . .   | 22 |
| The Basic Interpretive Cycle . . . . .                              | 22 |
| Arithmetic and Logical Operations . . . . .                         | 23 |
| Block Structure and Declarations . . . . .                          | 24 |
| Pointer Variables and Transfer of Data . . . . .                    | 27 |
| Transfers of Control Within a Program . . . . .                     | 29 |
| List Operations and String Processing . . . . .                     | 30 |
| Reusage of Discarded List Cells . . . . .                           | 33 |
| Communication of Data . . . . .                                     | 33 |
| Acknowledgements . . . . .  | 36 |
| Bibliography . . . . .  | 37 |
| Appendix 1. Features of the EULER Language . . . . .                | 38 |
| Appendix 2. SLIP Routines Used in Implementing EULER . . . . .      | 41 |
| Appendix 3. The EULER Translator Listing . . . . .                  | 43 |
| Appendix 4. The EULER Polish Interpreter Listing . . . . .          | 58 |
| ILLUSTRATION: Figure 1, A Portion of the EULER Translator . . . . . | 14 |



## Part I. The EULER Translator

### Introduction

In a previous paper (7), we introduced methods of designing programming language translators using a pushdown automaton model and a notation for specifying how the language of the translator is mapped into its translation. In that paper, a very small programming language was used in the design of a small translator system for purposes of demonstration. This paper will present a much expanded example of translator design based on the EULER programming language of Wirth and Weber (11). Much of the discussion in this paper will draw upon results and notation used in the previous paper.

Our first order of business is a syntax of EULER. Readers of the Wirth and Weber paper will note that the original EULER grammar was written in a very stylized fashion with many extra symbols introduced in order to fit the grammar into the framework of the Wirth and Weber translation algorithm. The EULER syntax to be presented in this report has been rewritten, not for the purpose of conforming to another translation algorithm, but to show that a more nearly transparent syntax can still yield an acceptable translator. In fact, the original EULER syntax would also "work" in our system, and the new version of the EULER syntax might be thought of as an exercise in aesthetics.

During the writing of the EULER translator system, an idea began to take shape concerning simple and clever methods for extending the EULER language and for using syntactic methods for the purpose of producing partially optimized code from the translator. These methods will be discussed in the following sections that deal with optimizations and methods of extending the system.

Readers interested in learning how to use the EULER language are referred to the extremely well written presentation in the original EULER paper (11). That paper contains examples of EULER programs, a few of which have typographic errors that the careful reader can use to test his knowledge of the language. Because the EULER report deals only with the reference language (i.e., had numerous symbols not available on many card-punch machines), an appendix is included with this paper to indicate the programming conventions used with the University of Maryland EULER system. This appendix might also be a useful first guide to EULER for programmers who know some ALGOL.

### An EULER Translation Grammar

The notation for the following grammar is an extension of the notation used in the previous report. In this notation, syntactic rules specifying the programming language are paired with rules of translation that specify how the language is translated, as in the following example:

| <u>Syntactic Rule</u>  | <u>Rule of Translation</u>                                 |
|--|--|
| $\langle \text{sum} \rangle \rightarrow \langle \text{term} \rangle$ | I  |
| $+\langle \text{term} \rangle$                                       | $\langle \text{term} \rangle$                              |
| $-\langle \text{term} \rangle$                                       | $\langle \text{term} \rangle.\text{NEG.}$                  |
| $\langle \text{sum} \rangle + \langle \text{term} \rangle$           | $\langle \text{sum} \rangle \langle \text{term} \rangle +$ |
| $\langle \text{sum} \rangle - \langle \text{term} \rangle$           | $\langle \text{sum} \rangle \langle \text{term} \rangle -$ |

In this example, sequences of symbols are rearranged in the process of translation. Thus, the infix addition of " $\langle \text{sum} \rangle + \langle \text{term} \rangle$ " is translated into the reverse Polish addition of " $\langle \text{sum} \rangle \langle \text{term} \rangle +$ ". The unary plus sign in the sequence " $+\langle \text{term} \rangle$ " is discarded in the translated sequence for  $\langle \text{term} \rangle$ . When the  $\langle \text{term} \rangle$  stands by itself as a descendant of a  $\langle \text{sum} \rangle$ , no symbols are added or deleted in the translation (hence, the notation used is I for Identity Translation).

In the rules of translation for the full grammar, there are certain sequences of symbols whose meaning may not seem obvious on a first reading. For example, to understand why the EULER statement

"A=B\*C+D.IN..,"

is translated into the sequence

".VRBL.A .VRBL.B .IN. .VRBL.C .IN.\* .VRBL.D.IN..IN.+=",

it is well to know something about how pointers to data are manipulated in the EULER reverse POLISH string interpreter. This interpreter is described in the next section. To understand why the EULER statement

".IF.A.AND.B..THEN.C.ELSE.D.,"

is translated into the sequence

".VRBL.A.IN..FAND..VRBL.B.IN..SAND..IF..VRBL.C.IN..SWCH..THEN..NOT..IF..  
VRBL.D..IN..SWCH..THEN....,"

it is well to read the following section on syntactic methods of producing optimized translations as well as the interpreter description. The descriptions of EULER programming in this paper and in the Wirth and Weber article will also be quite useful.



The full EULER grammar follows. A comparison of this grammar with the original one will satisfy the interested reader that the languages are virtually identical, although the grammars are different in form.

### The Grammar

#### Syntactic Rule

```

<program>→.ENTRY<block>.EXIT.
<block>→<blockhead><body>.END
<blockhead>→.BGIN.
    |<blockhead><labeldec>.,
    |<blockhead><vardec>.,
<vardec>→.NEW.<name>
    |<vardec>,<name>
<labeldec>→.LABL.<name>
    |<labeldec>,<name>
<body>→<body>.,<stat>
    |<stat>
<stat>→<labeldef><stat>
    |<expr>
<labeldef>→<name>..
<expr>→<block>
    |<disj>
    |<var>=<expr>
    |.GOTO.<prim>
    |.OUT.<prim>
    |<condition><consequence><alternative>
<condition>→.IF.<expr>
<consequence>→.THEN.<expr>
<alternative>→.ELSE.<expr>
<disj>→<conj>
    |<disj>.OR.<conj>
<conj>→<neg>
    |<conj>.AND.<neg>
<neg>→<relation>

```

#### Rule of Translation

```

<block>
    I
    I
    <blockhead><labeldec>
    <blockhead><vardec>
    .NEW.<name>
    <vardec>.NEW.<name>
    .LABL.<name>
    <labeldec>.LABL.<name>
    I
    I
    I
    I
    .LBDF.<name>
    I
    I
    <var><expr>=
    <prim>.GOTO.
    <prim>.OUT.
    I
    <expr>.IF.
    <expr>.SWCH..THEN..NOT..IF.
    <expr>.SWCH..THEN..,
    I
    <disj>.FSOR.<conj>.SCOR.
    I
    <conj>.FAND.<neg>.SAND.
    I

```

Syntactic Rule

|.NOT.<relation>  
 <relation>→<sum>  
 |<sum><sup>(1)</sup><relop><sum><sup>(2)</sup>  
 <relop>→{.EQ.|.NEQ.|  
 .GT.|.GEQ.|.LT.|.LEQ.}  
 <sum>→<term>  
 |+<term>  
 |-<term>  
 |<sum>{+|-}<term>  
 <term>→<factor>  
 |<term>{\*|/|./.|.MDLO.}<factor>  
 <factor>→<catena>  
 |<factor>\*\*<catena>  
 <catena>→<prim>  
 |<catena>.CNCT.<prim>  
 <prim>→.UNDF.  
 |<var>  
 |<label>  
 |(<expr>)  
 |<procdef>  
 |<referenceprim>  
 |<listprim>  
 |<numberprim>  
 |<logicalprim>  
 |.TAIL.<prim>  
 |<var><list>  
 |<symbolprim>  
 <label>→<name>  
 <var>→<name>  
 |<var>.IN.  
 |<var>(<sum>)  
 <referenceprim>→.AT.<var>  
 <listprim>→<list>

Rule of Translation

<relation>.NOT.  
 I  
 <sum><sup>(1)</sup><sum><sup>(2)</sup><relop>  
 I  
 I  
 <term>  
 <term>.NEG.  
 <sum><term>{+|-}  
 I  
 <term><factor>{\*|/|./.|.MDLO.}  
 I  
 <factor><catena>\*\*  
 I  
 <catena><prim>.CNCT.  
 I  
 <var>.IN.  
 <label>.IN.  
 <expr>  
 I  
 I  
 I  
 I  
 I  
 <prim>.TAIL.  
 <var>.LSCL.<list>  
 I  
 .VRBL.<name>  
 .VRBL.<name>  
 <var>.IN.  
 <var><sum>  
 <var>  
 I



Syntactic Rule

| .LIST.<sum>  
 <list>→<listhead>).  
 | <listhead><expr>).  
 <listhead>→.(  
 <listhead><expr>,  
 <numberprim>→<number>  
 | .REAL.<disj>  
 | .LNGT.<catena>  
 | .ABST.<sum>  
 | .INTG.<sum>  
 <logicalprim>→.TRUE.  
 | .FLSE.  
 | .LGCL.<sum>  
 | <typeinquiry><var>  
 <typeinquiry>→{.ISNU.|  
 .ISLO.|.ISLA.|.ISLI.|.ISPR.|  
 .ISRE.|.ISSY.|.ISUN.}  
 <symbolprim>→.\* <6-symbolstring>  
 <procdef>→<prochead><expr>\$.  
 <prochead>→\$.  
 | <prochead><formaldec>.,  
 <formaldec>→.FRML.<name>  
 | <formaldec>,<name>  
 <6-symbolstring>→{<letter>|<digit>|<blank>  
 |,|.|\$|\*|?|=|+|-|)|}^6  
 (i.e., 6 characters)  
 <name>→{<letter>{<letter>|<digit>}^k,  
 k=0,1,...,5}  
 (i.e., a name is a letter followed  
 by up to five letters or digits.)  
 <number>→<integer>  
 | <integer>^(1) .<integer>^(2)  
 <integer>→<digit>

Rule of Translation

<sum>.LIST.  
 I  
 I  
 I  
 I  
 .NMBR.<number>  
 <disj>.REAL.  
 <catena>.LNGT.  
 <sum>.ABST.  
 <sum>.INTG.  
 I  
 I  
 <sum>.LGCL.  
 <var><typeinquiry>  
 I  
 I  
 I  
 I  
 <prochead><formaldec>  
 .FRML.<name>  
 <formaldec>.FRML.<name>  
 I  
 I  
 TEMP(1)  
 TEMP(1)+SCALE\*TEMP(2)  
 SCALE←0.1;  
 TEMP(i)←value.<digit>.;

Syntactic Rule

|<integer><digit>

<digit>→{0|1|...|9}

<letter>→{A|B|...|Z}

Rule of Translation

TEMP(i)←10×TEMP(i)+value.( digit ).;

SCALE 0.1×SCALE;

I

I



### Syntactic Methods for Optimizing Expressions

At two points in the grammar above, the rules of translation specify translated sequences of symbols that are not in reverse Polish. The motivation for doing this in the grammar is to provide a framework for executing the resulting expressions so as to skip over redundant portions of the translated string. One of the instances of this translated structure is the rule

`<disj>→<disj>.OR.<conj>.`

In ordinary Polish notation, the rule of translation would be the sequence

`<disj><conj>.OR.`

However, this translation does not take into account the fact that, if the `<disj>` is `.TRUE.`, there is no need to evaluate the `<conj>` in the expression.

It was decided instead to write the rule of translation as follows:

`<disj>.FSOR.<conj>.SCOR.`

When this translated expression is executed, the `.FSOR.` operator is read immediately after the `<disj>` portion of the expression is evaluated. The effect of executing `.FSOR.` is to cause the interpreter to skip over the `<conj>` when the `<disj>` is evaluated as `.TRUE.`. In order to skip over the segment of program between `.FSOR.` and `.SCOR.`, the interpreter routine treats the program text as though it were a table of labels, and scans ahead until it encounters an `.SCOR.` that matches the `.FSOR.` by which the routine was activated. Thus, the `.SCOR.` operator is not executed by the interpreter, but is used as a place marker in the translated program.

The syntactic rule `<conj>→<conj>.AND.<neg>`

is matched with the translation rule

`<conj>.FAND.<neg>.SAND.`

for the same reason as given above. In this case, if `<conj>` has logical value `.FALSE.`, the translated program segment between `.FAND.` and `.SAND.` is skipped over by the interpreter routine.

A similar rationale was used in determining the rules of translation used for conditional statements. In this case, it was decided to break down EULER conditional statements from the form

`".IF.<expr>(1).THEN.<expr>(2).ELSE.<expr>(3).,"`

into the form `".IF.<expr>(1).THEN.<expr>(2)`

`.IF..NOT.<expr>(1).THEN.<expr>(3).,"`

In order to avoid evaluating  $\langle \text{expr} \rangle^{(1)}$  twice in the translated program, the following strategy was developed:

At run time, the EULER Polish interpreter stores the values of evaluated expressions in sequence on top of a stack of operands. It is therefore possible to retain the value of  $\langle \text{expr} \rangle^{(1)}$  for re-use in the translated program. Thus, the actual translation of the conditional statement above is the following sequence:

" $\langle \text{expr} \rangle^{(1)}$  .IF. $\langle \text{expr} \rangle^{(2)}$  .SWCH..THEN..NOT..IF. $\langle \text{expr} \rangle^{(3)}$  .SWCH..THEN...,"

Here, the .IF. command leaves the value of  $\langle \text{expr} \rangle^{(1)}$  in place on the operand stack. If  $\langle \text{expr} \rangle^{(1)}$  is true,  $\langle \text{expr} \rangle^{(2)}$  is evaluated, and, by the EULER conventions,  $\langle \text{expr} \rangle^{(2)}$  will also reduce to a single value. When  $\langle \text{expr} \rangle^{(2)}$  is completed, the operand stack will contain the sequence

value.( $\langle \text{expr} \rangle^{(1)}$ )., value.( $\langle \text{expr} \rangle^{(2)}$ ).

as the two topmost operands. It is then only necessary to switch these two operands (using the .SWCH. command), negate the topmost operand (using .NOT.), and skip over  $\langle \text{expr} \rangle^{(3)}$  to the .THEN. that matches the .IF. operator. The extra semicolon after the .THEN. place marker serves to unstack the value of .NOT. $\langle \text{expr} \rangle^{(1)}$  at the end of the conditional statement. Thus, in this case also, the use of a placemaker known to the interpreter facilitates skipping over redundant portions of the translated program.

### Some Syntactic Methods of Extending the EULER Language

After developing the appropriate techniques for breaking down conditional statements and for optimizing logical expressions, the next question concerns using these syntactic tricks to provide extended facilities in the EULER language. The extensions to be described in what follows have not yet been programmed into the system, there being a minor obstacle of time and money presently obstructing progress.

The introduction of full string-processing facilities into the EULER system is the first example to be considered. Without altering the EULER interpreter, and with very little reprogramming of the translator, we can effect the following

improvement:

| <u>Syntactic Rule</u>  | <u>Rule of Translation</u>                           |
|--|--|
| <code>&lt;prim&gt; → &lt;stringprim&gt;</code>   | I  |
| <code>&lt;stringprim&gt; → &lt;stringhead&gt;/.<br/>    &lt;stringhead&gt; → ./</code> | <code>&lt;stringhead&gt;).</code><br><code>.(</code> |
| <code>  &lt;stringhead&gt;&lt;symbol&gt;</code>  | <code>&lt;stringhead&gt;.*&lt;symbol&gt; ,</code>    |

Here, a string of arbitrary length is translated into a list whose cells store the symbols of the string one symbol to the cell in sequence. With this arrangement, it is possible to manipulate strings using the list concatenation operator, the .TAIL. operator, and using EULER subroutines for performing tests for list equality and containment.

The second example involves the addition of facilities for reading in data at run time within the framework of the EULER system. In this case, additional structures must be provided in the EULER Polish string interpreter. These facilities would take the form of routines for converting numbers into their internal representation and for packing string data. The added syntax might resemble the following rule system:

| <u>Syntactic Rule</u>  | <u>Rule of Translation</u>   |
|--|--|
| <code>&lt;program&gt; → .ENTRY&lt;block&gt;.EXIT.<br/>      .ENTRY&lt;data&gt;.,&lt;block&gt;.EXIT.</code>   | <code>&lt;block&gt;</code><br><code>&lt;data&gt;&lt;block&gt;</code> |
| <code>&lt;data&gt; → &lt;datahead&gt;.END.</code>  | I  |
| <code>&lt;datahead&gt; → .DATA.&lt;item&gt;<br/>      &lt;datahead&gt;.,&lt;item&gt;</code>  | I<br>I   |
| <code>&lt;item&gt; → &lt;number&gt;<br/>      &lt;stringprim&gt;<br/>      &lt;datalist&gt;</code>   | I<br>I<br>I  |
| <code>&lt;datalist&gt; → &lt;datalisthead&gt;).<br/>    &lt;datalisthead&gt; → .(&lt;item&gt;<br/>          &lt;datalisthead&gt;,&lt;item&gt;</code> | I<br>I<br>I  |

With this program structure, the <data> portion could be read in by a subroutine that leaves the <data> in a pre-arranged location in memory. The interpreter routine could then be read in on top of the data subroutine, and the translated



program would be executed. A statement of the form ".READ. prim " would then store an appropriate link to some segment of the read-in data on top of the operand stack of the interpreter.

The third example involves the use of a syntactic notation to expand the EULER Language into a self-extendible programming language similar to MAD/1 (4) and ALGOL 68 (10). By an extendible programming language, people currently mean the following two things:

- (a) A language in which the programmer can specify new data types and data structures composed of novel configurations of data elements.
- (b) A language in which the programmer is able to reorder the priorities of expression operators and is able to specify arbitrary new operations at will.

In EULER, there already exists a very general mechanism for allowing programmers to manipulate data structures, namely, the list mechanism. EULER lists can be constructed from arbitrary combinations of data elements. However, EULER only has eight data types with no facilities for extending their ranges. Such range-extension facilities depend on the machine on which the language is implemented, and algorithms for specifying such things as numbers of arbitrary precision must be written for the machine in question. Hence, our example will concentrate on the machine-independent problem of specifying new operators in programs.

Any reasonable programming language must presuppose the existence of a standard set of expression operators before provision is made for allowing programs to expand this set of operators. With each standard operator will be associated a standard precedence level, and the operators to be introduced by the programmer must also have precedence levels. As the term is currently used, operator precedence (or priority) is a measure of how expression operators compare in binding power. For example, exponentiation is said to have a lower precedence than addition, because exponentiation is performed before addition in arithmetic expressions. Thus, precedence imposes on the operations of a language. This ordering is reflected in the ordering of syntax rules in programming language grammars. In the EULER grammar above, rules are ordered so that list concatenation is performed first, then exponentiation, and so on, until the operation of value assignment. From concatenation to assignment of value there are nine levels of precedence.

Our approach in providing for the programming of new operations is to assign these operations to one of nine classes of operators, reflecting the nine levels in the original grammar. We accomplish this by adding an operator declaration to the language, and by permitting the programmer to define new operators and rearrange their precedences at will. The rules for permitting this in our system are as follows:

| <u>Syntactic Rule</u>   | <u>Rule of Translation</u>  |
|---|---|
| <code>&lt;expr&gt;→&lt;var&gt;&lt;opname&gt;&lt;expr&gt;</code><br> <disj>  | <code>.(&lt;var&gt;,\$&lt;expr&gt;\$.)..CALL.&lt;opname&gt;</code><br>I         |
| <code>&lt;disj&gt;→&lt;disj&gt;&lt;opname&gt;&lt;conj&gt;</code><br> <conj> | <code>.(.\$&lt;disj&gt;\$.,.\$&lt;conj&gt;\$.)..CALL.&lt;opname&gt;</code><br>I |
| :   | :   |
| <code>&lt;catena&gt;→&lt;catena&gt;&lt;opname&gt;&lt;prim&gt;</code>        | <code>.(.\$&lt;catena&gt;,\$&lt;prim&gt;\$.)..CALL.&lt;opname&gt;</code>        |

|   |  |
|---|--|
| <code>&lt;blockhead&gt;→</code><br><code>&lt;blockhead&gt;&lt;operatordec&gt;.,</code><br><code>&lt;operatordec&gt;→.OPRT.&lt;opname&gt;</code><br>  <code>&lt;operatordec&gt;,&lt;opname&gt;</code><br>(Programs now have operator, data variable, and label declarations.<br>Procedures may then have operators global to them, but may not have<br>operators passed to them as parameters of procedure calls.) | <code>&lt;blockhead&gt;&lt;operatordec&gt;</code><br><code>.OPRT.&lt;opname&gt;</code><br><code>&lt;operatordec&gt;.OPRT.&lt;opname&gt;</code> |
|---|--|

| <u>Syntactic Rule</u>   | <u>Rule of Translation</u>                    |
|---|---|
| <code>&lt;numberprim&gt;→.PREC.&lt;opname&gt;</code><br>(.PREC. inquires about the<br>precedence of the following<br><opname>.) | I   |
| <code>&lt;expr&gt;→&lt;opname&gt;=&lt;opdef&gt;</code><br>(Definition of new operators<br>and redefinition of old ones.)        | <code>.OP.&lt;opname&gt;&lt;opdef&gt;=</code> |

Syntactic RuleRule of Translation

|   |   |
|---|---|
| <code>&lt;opdef&gt; → &lt;defhead&gt;&lt;expr&gt;\$.</code>   | I   |
| <code>&lt;defhead&gt; → &lt;formalpart&gt;&lt;precedencepart&gt;</code>   | I   |
| <code>&lt;formalpart&gt; → \$.FRML.&lt;name&gt;.,</code><br><code>           . \$.FRML.&lt;name&gt;,&lt;name&gt;.,</code>   | <code>\$.FRML.&lt;name&gt;</code><br><code>\$.FRML.&lt;name&gt;.FRML.&lt;name&gt;.</code> |
| <code>&lt;precedencepart&gt; → RANK=&lt;digit&gt;.,</code>  | <code>RANK.NMBR.&lt;digit&gt;=.,</code>   |
| <code>&lt;opname&gt; → {= + - * ... .CNCT.</code><br><code>           . {&lt;letter&gt; &lt;digit&gt;}<sup>4</sup>.}</code> | I   |

In the syntax above, the `<opname>` in each rule is translated into a procedure call, with parameters consisting of the operands associated with each `<opname>`. These procedure calls either refer to the "standard" operator associated with a particular precedence level or refer to the translated `<opdef>`'s introduced into the program. It is assumed that the translator will automatically enclose each translated program with an extra outer block containing the procedure definitions of the standard operators. In this way, the standard operators can be redefined within a particular program, but will regain their usual meaning upon exit from the block in which the redefining statement occurred.

A certain amount of optimization is still possible within the framework of this extendible translator. As an example, suppose we write the following routine as the procedure corresponding to the standard operator for logical conjunction:

```
.AND.= $.FRML.X,Y.,RANK=7.,
      .BGIN..NEW.Z.,Z=X.,
      .IF.Z.THEN.
          .IF.ISLO.Y.THEN.Y.ELSE..GOTO.ERROR
      .ELSE. Z .END.$.
```

The actual parameters in the procedure call for `.AND.` are subexpressions surrounded by `$.` and `$.` . Thus, the effect of the statement

```
"Z=X.,"
```

is to evaluate the X parameter only once and not to evaluate the Y parameter unless Z is `.TRUE.` .

Other possible extensions to the EULER language include the addition of ALGOL-like iteration statements, as well as more rapid FORTRAN-style iterations. In both these cases, the additions can be made easily by incorporating a little extra syntax into the translator, and by having the language of that syntax translate into appropriate procedure calls to global "system" procedures.

#### How a Section of the Translator Was Designed--An Example

It is assumed that readers of this section will have some familiarity with the translator example in the previous paper (8) on this subject. In order to simplify the programming of the translator, it was decided to have the reserved words of the language perform as many functions as possible in the translation. Thus, the reserved words actually appear in translations as commands for the interpretive system where appropriate, and are stored on the pushdown store of the translator in place of the "nonterminal symbols" of the normal-form version of the grammar. For example, in the normal-form grammar for EULER, the rule

$$\begin{aligned} \langle \text{expr} \rangle &\rightarrow X_1 \langle \text{alternative} \rangle \\ X_1 &\rightarrow X_2 \langle \text{consequence} \rangle \\ X_2 &\rightarrow \langle \text{condition} \rangle \end{aligned}$$

By letting  $X_1$  be .THEN. and  $X_2$  be .IF. in the translator, the coding is greatly simplified, and no ambiguities are introduced, since the  $X_i$  can be treated as "new and distinct" symbols. The flowchart of Figure 1, showing the transitions to and from the box corresponding to  $\langle \text{expr} \rangle$ , illustrates how the EULER translator given in Appendix was programmed.

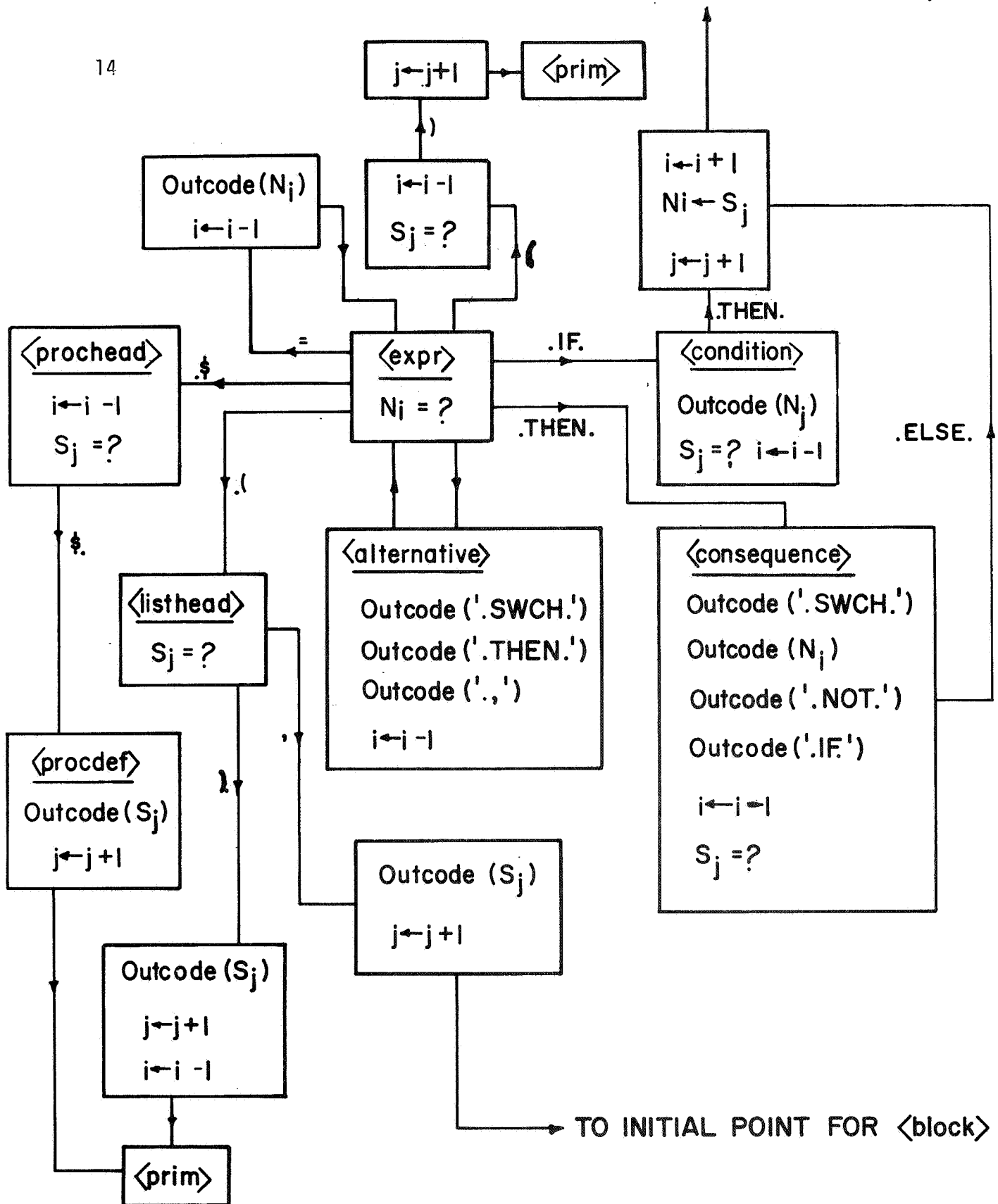


Figure 1. A Portion of the EULER Translator



## Error Messages and Error Scans in the Translator

In the flowchart segment of Figure 1, no error exits are indicated. This is because, with this translator-design algorithm, error exits are implied; i.e., when the appropriate combination of program and pushdown-store symbols are not read at some stage of the computation, a syntactic error in the program is indicated. Rather than halt the processing of an input program at this point, it was decided to provide a full error scan, one that points out the location of each error as it occurs.

The error-scanning mechanism is based on the following observation concerning the EULER syntax: The <body> of a program consists of sequences of the form

<body>.,<stat>

where the <stat> may be some additional sequence of statements surrounded by .BGIN. and .END. . Thus, it is possible to treat this level of the programming language as a synchronizable encoding (7) in which the decoding process can be resumed after an error occurs because the decoding algorithm does not "know" the difference between an input sequence of the form

"<stat><sup>(1)</sup>.,<stat><sup>(2)</sup>.,<stat><sup>(3)</sup>.,"

and one of the form

"<stat><sup>(1)</sup>.,<stat><sup>(3)</sup>.,".

Because of this property of the grammar, a scanning routine can be constructed so that, whenever the translator finds a program error, the routine scans forward in the program until it encounters the first semicolon or .END. following the error. This routine then resets the translator so that translation resumes at the program location following the semicolon or .END. . If a semicolon is encountered, the translator resumes translation at its initial point; if an .END. is encountered, the translator resumes translation in the section corresponding to <body>.

It is conjectured here, but not proved, that this error-scanning mechanism will never permit the printing out of a translated program containing an error. The reason for this conjecture is that the syntax-checking mechanism of the translator is critically sensitive to the balancing of .BGIN.'s and .END.'s, left-and right parentheses, etc., in a program. Thus, when an error is discovered in a program, some extra symbol will remain on the pushdown store, and this symbol will ultimately cause an error message when

the .EXIT. command of the program is read. With an error message at this point in the program, it is a very simple matter to suppress the punching out of the translated program. So far, experience with the translator has borne out this conjecture.

### A Missing Feature of the Translator

In section 4.7.3.3 of the revised ALGOL report (14), the following description of procedure calls appears:

...If the procedure is called from a place outside the scope of any nonlocal quantity of the procedure body the conflicts between the identifiers inserted through this process of body replacement and the identifiers whose declarations are valid at the place of the procedure statement...will be avoided through suitable systematic changes of the latter identifiers.

The meaning of this condition can be seen in terms of the following EULER programming example:

```
.ENTRY.BGIN..NEW.A,B.,
    A=2.,
    B=.$A$.,
    .BGIN. .NEW.A.,A=5.,
    .OUT.B.END.,
    .OUT.B .END. .EXIT.
```

Following the dictates of the ALGOL report, the number 2 would be written out twice by the program above. This is because the variable A is global to the definition of procedure B, and hence, some translation routine would have to rename A so that the translated program would be similar to what would appear if the above program had originally been written in the following way:

```
.ENTRY .BGIN. .NEW.Z,B.,
    Z=2.,
    B=.$Z$.,
    .BGIN..NEW.A.,A=3.,
    .OUT.B.END.,
    .OUT.B.END..EXIT.
```

Although this ALGOL principle is not directly stated in the EULER report, there is an indication that variables in procedures are bound to the block in which they occur. Thus, something like section 4.7.3.3 must have been part of the original EULER.

In our EULER translator, the effect of executing the program example above would be to write out the number 5 followed by the number 2. Thus, section 4.7.3.3 has not yet been implemented for our system, although the programming involved is fairly straightforward. At present, the second program above must be submitted to obtain the correct results, and so the programmer must rename his program variables rather than have the translator perform the renaming.

#### Translation Times and Memory Requirements

The present EULER translator programmed in FORTRAN IV on the IBM 7094 machine takes approximately 5600 words of memory, including FORTRAN system routines and space for 500 packed words of EULER program to be translated. At present, 500 words of translated program output are obtained in about .06 minutes of execution time, and times for smaller programs are somewhat better, although difficult to measure.

## Part II. The EULER Interpretive System

### Introduction

This section consists of a description of how the EULER interpreter works. The structures used by the system and the commands that manipulate these structures are described, and algorithms written in a subset of EULER depict what happens when the commands are used. It is intended that these descriptions and algorithms, together with the actual FORTRAN IV version of the system given at the end of this report, will give the reader two somewhat different viewpoints of how the interpretive system functions.

This section of the report could also be considered an advanced manual for programming in the EULER reverse Polish language, although it is intended that the EULER translator be used for "writing" the programs used by the interpreter. It should be noted that the interpreter system now in use accepts programs written on punched cards, with the commands and data punched between columns 7 and 72 of the cards. Individual commands are punched left-justified into separate fields six columns in length. Thus, a total of eleven commands and data items can appear on one card. The EULER translator receives programs punched "free format" between columns 7 and 72 of cards and translates these programs into the interpreter notation, properly spaced and justified on punched cards.

### Structures within the EULER Polish Interpreter

The following are the stacks and computer words used by the processor that executes the translated EULER code:

|          |  |
|----------|--|
| CODE -   | The array in which the translated EULER program is stored.   |
| I -      | The index of CODE, pointing to the translated instruction currently being executed.                                    |
| IDNTLS - | The table of names, types and values used to implement dynamic storage allocation of variables.                        |
| T -      | The index of IDNTLS, pointing to the location of the most recently declared variable name in a program being executed. |
| OPRND -  | The stack of intermediate values and links stored in the process of executing a program.                               |
| IOPRND - | The stack of data types associated with the values and links stored in OPRND.  |
| IK -     | The index of OPRND and IOPRND.   |
| STORAJ - | The stack of locations in the IDNTLS name table where storage of names begins for each block.                          |
| BLKNUM - | The index of STORAJ that records the dynamic block number at each point in the program being executed.                 |
| JUMPBK - | The stack of procedure return addresses used in execution of recursive and nonrecursive procedure calls.               |
| PRAMLS - | The stack of pointers to parameter lists of procedures called during program execution.                                |
| JMPRTN - | The index of JUMPBK and PRAMLS, giving the level of nesting of procedure calls at any point in a program.              |
| LAVS -   | Array name used for setting up the pool of linked list cells used by the interpreter's list-processing routines.       |
| AVSL -   | The machine word containing pointers to the beginning and end of LAVS.   |
| LSTRTN - | A stack used for reading out the contents of lists that contain sublists during execution of a program.                |
| LSTCNT - | The index of LSTRTN, indicating the depth of the sublist currently being read using LSTRTN.                            |



The following description of the EULER interpreter commands consists largely of algorithms written in a subset of EULER. It was felt that these algorithms can be concisely written and are intelligible to anyone familiar with ALGOL, and possibly to some familiar with MAD or FORTRAN as well. Appendix 4 is the actual interpreter program, and is written in FORTRAN IV. By comparing the EULER algorithms with the corresponding FORTRAN, it will be noted that the function JUMPTO.(.\*DUMMY)., where .\*DUMMY is some chosen operator, corresponds to several slightly different FORTRAN routines used for scanning forward in the interpreted program without executing the segment of CODE that is thereby bypassed.

#### Typing and Storage of Data

Data appears in IDNTLS, the interpreter name table, as triples of Computer words. The first word in each triple is the program-declared name, the second word contains a data type code, and the third word contains the actual datum. Since the index of IDNTLS is T, this means that IDNTLS(3\*T-2) is an alphanumeric word, IDNTLS(3\*T-1) contains the type, and IDNTLS(3\*T) contains the datum. Data is also stored on the operand stack OPRND. Here, OPRND(IK) denotes some value or linkage, and IOPRND(IK) is the integer type code of the datum in OPRND(IK).

The other important structures in which data can be stored and manipulated are list cells drawn from LAVS. Each list cell consists of a pair of adjacent computer memory words, in the manner of SLIP list cells ( 11 ). The SLIP conventions regarding fields within a computer word are used here: That is, a given word is divided into two address-sized fields, called the LNKL and LNKR fields, and a smaller ID field. The datum is stored in the second cell of each word pair, and the typing information for that datum appears in the ID field of the first cell of the pair. To maintain a semblance of consistency within the interpreter routines, each pair of words IDNTLS(3\*T-1) and IDNTLS(3\*T) is made to resemble a list cell in that the typing information for the datum in IDNTLS(3\*T) is stored in the ID field of IDNTLS(3\*T-1). In Appendix 2 is a brief description of the SLIP routines used in this paper to describe EULER.

At present, there are eight data types manipulated by the EULER system: numerical constants and variables, logical constants and variables, labels, lists, references or pointers, procedures, alphanumeric symbols, and a last category of undefined data. The type coding mechanism works as follows: If the ID field of IDNTLS(3\*T-1) is all zero, or if IOPRND(IK) is an integer zero, the associated datum is a floating-point number. If the code is an integer one, the associated datum is logical. If the code is two, the datum is a label whose location in the CODE array is stored as a floating point number. If the ID code is three, the associated datum is an integer link to a list stored in the LNK R field of the datum word. If the ID code is four, the associated datum is called undefined. If the ID code is five, the associated datum is an integer pointer to some other datum. This link is likewise stored in the LNK R field, and is a reference to either the first word of a list cell or to IDNTLS(3\*T'-1) for some T'. If the ID code is six, the datum is the location of the first command of a procedure somewhere in the CODE array, and this location is stored in floating point. Finally, if the ID code is seven, the corresponding datum contains six alphanumeric symbols, including blanks (for the IBM 7094 machine).

The following operators exist for testing the type of an operand during execution of a program. These operators check the type code of the topmost operand on the OPRND stack.

.ISNU.(isnumber) - Changes OPRND(IK) to true if IOPRND(IK) is 0. Otherwise OPRND(IK) becomes false. IOPRND(IK) becomes 1 (logical type).

.ISLO.(islogical)

.ISLA.(islabel)

.ISLI.(islist)

.ISUN.(isundefined)                      All defined analogously to .ISNU.

.ISRE.(isreference)

.ISPR.(isprocedure)

.ISSY.(issymbol)

The following two operators exist to perform type conversions between numbers and logical values. They permit a certain flexibility in the use of arithmetic expressions or logical expressions in "unorthodox" computations:

|                                 |   |
|---------------------------------|---|
| .LGCL.(logical type conversion) | LOGCAL: .IF.IOPRND(IK).NEQ.0.THEN.<br>.GOTO.ERROR .ELSE.IOPRND(IK)=1;.GOTO.<br>CYCLE; |
| .REAL.(number type conversion)  | TOREAL: .IF.IOPRND(IK).NEQ.1.THEN..GOTO.<br>ERROR.ELSE.IOPRND(IK)=0;.GOTO.CYCLE;      |

### Insertion of Data

At the present time, the interpreter does not use tables of numerical and logical constants. Instead, such data is passed to the interpreter at each point of occurrence in the translated program. Numbers are passed in the form of octal representations of floating point numbers, and are stored in two consecutive six-character words. The operator .NMBR. signals to the interpreter that the following two words in CODE are a number, and a machine-language subroutine packs this number into OPRND(IK), after IK has first been incremented and IOPRND(IK) set to zero. Another operator .\* signals to the interpreter that the following word in CODE consists of six alphanumeric characters. Index IK is incremented, IOPRND(IK) is set to 7, and the six-character word is stored in OPRND(IK).

Logical truth and falsity are presented to the interpreter as .TRUE. and .FLSE. respectively. For each of these operands, IK is first incremented and IOPRND(IK) set to 1. If the operand is .TRUE., OPRND(IK) is set to 1, otherwise 0. Finally, it is possible to insert .UNDF. (an undefined constant) on the operand stack. In this case also, IK is incremented. IOPRND(IK) is then set to 4, and OPRND(IK) is set to zero.

### The Basic Interpretive Cycle

Several of the basic interpreter commands have already been discussed. These commands and the remaining ones to be presented in this report are all executed by a basic interpretive cycle that reads commands from the translated EULER program and passes control to routines that execute these commands. This interpretive cycle can be described in the following steps:

- I. Initialization of interpreter indices:  
 I=0; IK=0; BLKNUM=0; T=0; JMPRTN=0;
- II. Creation of the LAVS pool of linked list cells:  
 INITAS.(.AT.AVSL,.AT.LAVS,2000).;
- III. Reading of the next command of the translated EULER program:  
 CYCLE: I=I+1; ~~EXECUTE~~.(CODE(I)).; .GOTO.CYCLE;

## Arithmetic and Logical Operations in the Interpreter

The following segmented table indicates in the left column the operator in CODE(I) to be executed and in the right column the corresponding interpretation algorithm:

### I. Unary Operators

|                                 |  |  |
|---------------------------------|--|--|
| .NEG.                           | NEGATE: .IF. IOPRND(IK).NEQ.0.THEN.        |  |
| (unary minus sign)              | .GOTO. ERROR .ELSE. OPRND(IK))=-OPRND(IK); |  |
|                                 | .GOTO.CYCLE;                               |  |
| .ABST. (Take absolute value)    | }  | Defined analogously to unary negation. |
| .INTG. (Truncate to an integer) |  |  |
| .NOT.                           | NOT:.IF. IOPRND(IK) .NEQ.1.THEN..GOTO.     |  |
| (logical negation)              | ERROR.ELSE.OPRND(IK)=1-OPRND(IK);          |  |
|                                 | .GOTO. CYCLE;                              |  |

### II. Binary Operators

|                             |  |   |
|-----------------------------|--|---|
| +(addition)                 | ADD:.IF. IOPRND(IK).NEQ.0 .OR. IOPRND(IK-1)      |   |
|                             | .NEQ. 0 .THEN. .GOTO. ERROR .ELSE.               |   |
|                             | .BGIN. OPRND(IK-1) = OPRND(IK-1)+OPRND(IK);      |   |
|                             | IK=IK-1; .GOTO. CYCLE .END.                      |   |
| - (subtraction)             | }  | Defined analogously to addition.              |
| * (multiplication)          |  |   |
| / (division)                |  |   |
| ./ (integer division)       |  |   |
| .MDLO. (modulo)             |  |   |
| ** (exponentiation)         |  |   |
| .EQ. (test for equality)    | ISSAME:.IF. IOPRND(IK) .NEQ. 0 .OR. IOPRND(IK-1) |   |
|                             | .NEQ. 0 .THEN. .GOTO. ERROR .ELSE.               |   |
|                             | .BGIN. OPRND(IK-1)= .IF. OPRND(IK-1)             |   |
|                             | .EQ. OPRND(IK) .THEN. 1.0.ELSE.0.0.END.          |   |
|                             | ;IK=IK-1; IOPRND(IK) =1; .GOTO. CYCLE;           |   |
| .NEQ. (test for inequality) | }  | Defined analogously to the test for equality. |
| .GT. (test for >)           |  |   |
| .LT. (test for <)           |  |   |
| .GEQ. (test for ≥)          |  |   |
| .LEQ. (test for ≤)          |  |   |

### III. Logical Operators Modified for Increased Execution Speed

|  |  |
|--|--|
| .FAND. (logical conjunction)<br>(Note: This is an optimized conjunction that is executed like an ALGOL conditional statement.) | .IF. IOPRND(IK) .NEQ.1.THEN.<br>.GOTO. ERROR .ELSE..IF.OPRND(IK)<br>.EQ. 0 .THEN.<br>I=JUMPTO.(.*.SAND.)..ELSE.<br>IK=IK-1;.GOTO.CYCLE;    |
| .FSOR. (logical disjunction)<br>(Note: This is an optimized disjunction that is executed like an ALGOL conditional statement.) | .IF. IOPRND(IK) .NEQ.1 .THEN.<br>.GOTO. ERROR .ELSE. .IF.<br>OPRND(IK) .EQ.1 .THEN.<br>I=JUMPTO.(.*.SCOR.)..ELSE.<br>IK=IK-1; .GOTO.CYCLE; |
| .SAND. (end label of a logical conjunction)  | .IF. IOPRND(IK) .NEQ.1<br><br>.THEN..GOTO.ERROR<br>.ELSE..GOTO.CYCLE;  |
| .SCOR. (end label of a logical disjunction)  |  |
| .IF. (the test operator in a conditional statement.)   | .IF.IOPRND(IK).NEQ.1.THEN.<br>.GOTO.ERROR.ELSE..IF.OPRND(IK)<br>.EQ.0.THEN.I=<br>JUMPTO.(.*.THEN.)..ELSE..UNDF.;<br>.GOTO.CYCLE;           |
| .THEN.   | .GOTO. CYCLE;  |
| .SWCH. (Exchange the two topmost operands during a conditional statement having an alternative.)                               | DATUM=OPRND(IK);TYPE=<br>IOPRND(IK); OPRND(IK)=OPRND(IK-1);<br>IOPRND(IK)=IOPRND(IK-1);<br>OPRND(IK-1)=DATUM;<br>IOPRND(IK-1)=TYPE;        |

### Block Structure and Declarations

Since EULER is a language having nested block structure and requiring that all program variables and labels be declared, appropriate commands exist for executing these functions. For the most part, these commands manipulate the IDNTLS table of declared names and the STORAJ stack of pointers to where the names of each currently active block are stored.



In this EULER system, the JUMPBK stack is the mechanism that implements returns of control from subroutines being executed. The stacking of subroutine return addresses permits recursive procedure calls to be executed in this system at the same speed and efficiency as non-recursive procedure calls. If a procedure leaves a value on the operand stack, that value is not destroyed by the procedure return mechanism. Thus, no extra mechanisms are necessary to implement procedures that return function values, and both types of procedures are treated alike.

There are two sequences of commands that initiate procedure calls in the EULER system. The sequence

".VRBL.<name>.IN.",

where name corresponds to a procedure variable, is used for calling those procedures that have no formal parameters. The sequence

".VRBL.<name>.LSCL..(<paramlist>)."

is used for calling procedures that have formal parameters. The actual parameters in the procedure call are put onto a list, and, when the command ). is interpreted, the link to this list is put on top of the PRAMLS stack before control is transferred to that subroutine. A description of the .IN. command is given in the following section, .LSCL. is discussed in the section on transfers of control, and .( and ). are described in the section on list operations. The following list of commands and algorithms includes those commands that are interpreted when a procedure is executed by the system.

|   |   |
|---|---|
| .BGIN.<br>(beginning of a<br>program block)<br>.\$.<br>(beginning of a<br>procedure definition) | BEGIN:BLKNUM=BLKNUM+1;<br>STORAJ(BLKNUM)=T+1;<br>.GOTO.CYCLE:<br>PROCHD:IK=IK+1;OPRND(IK)=I;<br>IOPRND(IK)=6;<br>I=JUMPTO.(.*\$.      ).;<br>.GOTO.CYCLE;<br>NEW:I=I+1; T=T+1; IDNTLS(3*T-2)=<br>CODE(I);SETDIR.(4,0,BLKNUM,<br>IDNTLS(3*T-1)).;IDNTLS(3*T)=0;<br>.GOTO.CYCLE.<br>LABEL:I=I+1;T=T+1;IDNTLS(3*T-2)=<br>CODE(I);SETDIR.(2,0,BLKNUM,<br>IDNTLS(3*T-1)).;=<br>SETDIR.(0,BLKNUM,LOCATE.(CODE(I)).,<br>IDNTLS(3*T)).;.GOTO.CYCLE;<br>LABDEF:I=I+1;.GOTO.CYCLE;<br>FORMAL:I=I+1;T=T+1;IDNTLS(3*T-2)=<br>CODE(I);LINK=LNKR.(PRAMLS<br>(JUMPRTN)).;.IF.LINK.EQ.0.THEN.<br>.BGIN.IDNTLS(3*T)=0;<br>SETDIR.(4,0,0,IDNTLS(3*T-1)).<br>.END..ELSE..BGIN.TEMP=<br>ID.(INHALT.(LINK)).;.IF.TEMP.EQ.<br>2.OR.TEMP.EQ.5.THEN.<br>SETDIR.(-1,-1,LNKL.(INHALT.(LINK+1)).<br>,IDNTLS(3*T-1))..ELSE..UNDF.;<br>PRAMLS(JUMPRTN)=INHALT.(LINK).;<br>IDNTLS(3*T)=INHALT.(LINK+1).;<br>.GOTO.CYCLE;<br>SEMCLN:.IF.IOPRND(IK).EQ.3<br>.THEN..BGIN.LLNK=LNKR.(<br>OPRND(IK)).;.IF.LNKL.(INHALT.(<br>LLNK)).EQ.0.THEN.ERASE.(<br>LLNK)..ELSE..UNDF..END.<br>.ELSE..UNDF.;IK=IK-1;.GOTO.<br>CYCLE; |
|---|---|

```

.END.                                END:INDEX=STORAJ(BLKNUM);
(end of a                            .IF.T.GEQ.INDEX.THEN.
program block)                       LOOP:.BGIN..IF.ID.(IDNTLS(3*T-1)).
                                      .EQ.3.THEN.ERASE.(LNKR.(
                                      IDNTLS(3*T))..ELSE..UNDF.;
                                      T=T-1;.IF.T.GEQ.INDEX
                                      .THEN..GOTO.LOOP.ELSE..UNDF.
                                      .END..ELSE..UNDF.;BLKNUM=
                                      BLKNUM-1;.IF.BLKNUM.GT.O
                                      .THEN..GOTO.CYCLE.ELSE.
                                      .GOTO.HALT;
$.                                  PRCEND:I=JUMPBK(JMPRTN);
(end of a procedure)                JMPRTN=JMPRTN-1;.GOTO.END;

```

### Pointer Variables and Transfer of Data

In the EULER language, a limited use of indirect addressing exists in the form of variables of type reference. The value associated with a reference variable is either the machine address of  $IDNTLS(3*T'-1)$ , for some  $T'$ , or the machine address of the first word of some list cell. As an example, we could write the following fragment of a valid EULER program:

```

...A=2; V=.AT.A;
V.IN.=V.IN.+1....

```

Here, A is a variable of type number, and V becomes a reference to A. The last command in the fragment causes A to acquire a numerical value of 3.

On a statement-by-statement basis, the program fragment would have been translated into the following fragment: (Note that .AT. is not translated directly.)

```

.VRBL.A      .NMBR.202400000000=
.,   .VRBL.V      .VRBL.A      =
.,   .VRBL.V      .IN.   .VRBL.V
.IN.   .IN.   .NMBR.201400000000+

```

The effect of the command `.VRBL.` is to fetch a pointer to the variable name that follows it in the translated program. Thus, the effect of executing the sequence

```
.VRBL.V      .VRBL.A      =
```

is to take the pointer for A and store it in IDNTLS as the datum for V. The effect of the command `.IN.` is to replace a pointer to some datum by the datum itself. Hence, the use of

```
.VRBL.V      .IN.  .IN.
```

in the translated program. Here, the datum acquired by the first use of `.IN.` is itself a pointer to another datum. The second use of `.IN.` finally captures that indirectly referenced datum. Thus, the effect of an `.AT.` in a program is the suppression of an `.IN.` in the translation.

The following list of commands presents the algorithms that are used to interpret references and assignments of value in this version of EULER. Note that error exits occur whenever some variable global to a program block is assigned as a reference to a variable local to that block.

|                         |   |
|-------------------------|---|
| <code>.VRBL.</code>     | VARIABLE: I=I+1; IVRBL=CODE(I);           |
| (fetches a              | ITEMP=T; LOOP1: IF IDNTLS(3*T-2)          |
| pointer to a            | .NEQ. IVRBL. THEN .BGIN. ITEMP=ITEMP-1;   |
| variable name)          | .IF. ITEMP.EQ.0. THEN .GOTO. ERROR        |
|                         | .ELSE. .GOTO. LOOP1. END. .ELSE. .BGIN.   |
|                         | IK=IK+1; IOPRND(IK)=5;                    |
|                         | SETDIR.(0, LNK. (IDNTLS(3*ITEMP-1)).,     |
|                         | MADOV.(IDNTLS(3*ITEMP-1))., OPRND(IK)).;  |
|                         | .GOTO. CYCLE. END.                        |
| <code>.IN.</code>       | INSIDE: ITYPE=ID.(CONT.(LNKR.(            |
| (fetches the            | OPRND(IK))).); IOPRND(IK)=ITYPE;          |
| datum referenced        | .IF. ITYPE.NEQ.6. THEN .BGIN. OPRND(IK)=  |
| by a pointer; initiates | CONT.(LNKR.(OPRND(IK)).+1).; .GOTO. CYCLE |
| procedure calls.)       | .END. .ELSE. .BGIN. IK=IK-1; JMPRTN=      |
|                         | 1+JMPRTN; JUMPBK(JMPRTN)=I;               |
|                         | BLKNUM=1+BLKNUM;                          |
|                         | STORAJ(BLKNUM)=1+T;                       |
|                         | I=CONT.(LNKR.(OPRND(IK)).+1).;            |
|                         | .IF. CODE(I+1).EQ.*.FRML.. THEN .GOTO.    |

```

ERROR.ELSE..GOTO.CYCLE.END.
= ASSIGN:.IF.IOPRND(IK-1).NEQ.5.THEN.
.GOTO.ERROR.ELSE..BGIN.TYPE=IOPRND(IK);
LRNK=LNKR.(OPRND(IK)).;LINK=
LNKR.(OPRND(IK-1)).;.IF.TYPE.EQ.5.OR.
TYPE.EQ.3.THEN..IF.LNKL.(OPRND(IK)).
.GT.LNKL.(OPRND(IK-1))..THEN..GOTO.ERROR
.ELSE..UNDF..ELSE..IF.TYPE.EQ.3.AND.
LRNK.GT.0.THEN..BGIN.SETIND.(-1,
LNKL.(CONT.(LRNK).)+1,-1,LRNK).;
SETDIR.(0,0,-1,OPRND(IK))..END..ELSE..UNDF.
;.IF.ID.(CONT.(LINK).)..EQ.3.AND.LNKR.(
CONT.(LINK+1).)..GT.0.THEN.
ERASE.(CONT.(LINK+1).)..ELSE..UNDF.;
SETIND.(TYPE,-1,-1,LINK).;OPRND(IK)=
CONT(LINK+1);OPRND(IK-1)=OPRND(IK);
IOPRND(IK-1)=IOPRND(IK);IK=IK-1;.GOTO.CYCLE;

```

### Transfers of Control Within a Program

Transfers of control in a programming language are both implicit and explicit. An explicit transfer is caused by a .GOTO. command. The .LSCL. command is implicit, since it does not appear in EULER programs, but is used by the translator system for transferring control to subroutines that have parameters. Note that .LSCL. only serves to find the location in the program of the procedure. The actual transfer of control takes place when ). is read.

```

.GOTO. JUMP:.IF.IOPRND(IK).NEQ.2.THEN.
.GOTO.ERROR.ELSE..BGIN.I=LNKR.(
OPRND(IK)).-1;JMPRTN=JMPRTN
+LNKL.(OPRND(IK)).-BLKNUM;.IF.
JMPRTN.LT.0.THEN.JMPRTN=0.ELSE.
.UNDF.;.GOTO.CYCLE.END.

```



```

.LSCL.                                LSCALL:LINK=LNKR.(OPRND(IK)).;
                                      .IF.ID.(CONT.(LINK)).EQ.6.THEN.
                                      .BGIN.IOPRND(IK)=6;OPRND(IK)=
                                      CONT.(LINK+1).;.GOTO.CYCLE.END.
                                      .ELSE..GOTO.ERROR;

```

### List Operations and String Processing

The EULER system has facilities for constructing and concatenating lists, for taking the suffix of a list, and for finding subscripted list elements. Moreover, each list call can store data of any legal type, including procedure, reference, label, etc. Thus, it is possible to execute a program by simply naming the elements of a list of procedures in order. Or, we can obtain the effect of an ALGOL switch by constructing a list of labels and writing .GOTO. followed by a subscripted list reference. Since characters can be stored on lists, it is possible to compare long character strings by writing procedures for testing two lists for equality, containment, etc.

The command .LIST. constructs an empty list whose length is given by the numerical value of the topmost operand on the OPRND stack. The commands .( and ). construct a list out of data elements that occur between them separated by commas. The subscripting command ) is evaluated one subscript level at a time and takes its information from the two topmost elements of the OPRND stack. The concatenation operator .CNCT. creates a copy of the two lists whose pointers are the topmost operands of the OPRND stack. Finally, the suffix operator .TAIL. finds the link to a list on top of the OPRND stack and replaces that link by a link to the following cell of that list.

The following example illustrates a distinction that should be understood when programming EULER: If A and B are lists in a program, then the two statements

C=.(A,B).; and C=A.CNCT.B;

do not have the same effect. In the first case, C is a list of two elements which point to A and B as sublists. In the second case, C is a list consisting of a copy of A joined to a copy of B.

The following table of commands and algorithms further describes the results produced by the list manipulation operators:

.LIST.

.(

,

(stores topmost  
OPRND in a list  
cell and replaces  
this operand with  
a link to the cell.)

).

```

LIST: .IF. IOPRND(IK) .NE. 0. THEN.
      .GOTO. ERROR. ELSE. .BGIN. IOPRND(IK)
      =3; INDEX=OPRND(IK); .IF. INDEX
      .EQ. 0. THEN. .BGIN. OPRND(IK)=0;
      .GOTO. CYCLE. END. .ELSE. .UNDF.;
      IVRBL=NUCELL; .IF. INDEX
      .EQ. 1. THEN. .GOTO. OUTPUT. ELSE. LLCELL=
      IVRBL; LOOP2: INDEX=INDEX-1; .IF.
      INDEX.LT. 1. THEN. .GOTO. OUTPUT. ELSE.
      .BGIN. LRCELL=NUCELL; SETIND. (4,
      -1, LRCELL, LLCELL) .; LLCELL=LRCELL;
      .GOTO. LOOP2. END. ; OUTPUT: SETDIR. (0,
      0, IVRBL, OPRND(IK)) .;
      LISTHD: IK=IK+1; IOPRND(IK)=3;
      OPRND(IK)=. *. (      ; .GOTO. CYCLE;
      COMMA: IKK=IOPRND(IK); IJJ=LNKR. (
      OPRND(IK)) .;
      .IF. IKK.EQ. 3. AND. IJJ.GT. 0. THEN.
      SETIND. (-1, LNK. (CONT. (IJJ) .) .+1, -1, IJJ) .;
      .ELSE. .UNDF.; TEMP=NUCELL;
      SETIND. (IKK, 0, 0, ITEMP); STRIND. (
      OPRND(IK), ITEMP+1) .; OPRND(IK)=TEMP;
      IOPRND(IK)=3; .GOTO. CYCLE;
      LSTEND: .IF. OPRND(IK) .EQ. . *. (      . THEN.
      .BGIN. OPRND(IK)=0; .GOTO. CYCLE. END.
      .ELSE. EXECUTE. (COMMA) .; LOOP3: .IF.
      OPRND(IK-1) .NEQ. . *. (      . THEN. .BGIN.
      SETIND. (-1, 0, LNK. (OPRND(IK)) ., LNK.
      . (OPRND(IK-1)) .) .; IK=IK-1; .GOTO. LOOP3
      .END. .ELSE. .IF. OPRND(IK-1) .NEQ. 6. THEN.
      .GOTO. CYCLE. ELSE. .BGIN. JMPRTN=1+
      JMPRTN; JUMPBK(JMPRTN)=I; T=T+1;
      PRAMLS (JMPRTN)=IDNTLS(3*T)=OPRND(IK);
      SETDIR. (3, 1, 0, IDNTLS(3*T-1)) .; I=OPRND(IK-1;
      IK=IK-2; .GOTO. BEGIN;

```

```

)
(subscript operator)
    RSBCP:INDEX=OPRND(IK);IK=IK-1;
    .IF.INDEX.LEQ.0.THEN..GOTO.CYCLE
    .ELSE.LINK=LNKR.(CONT.(1+LNKR.(
    OPRND(IK)).)).;.IF.LINK.GT.0.THEN.
    .BGIN.LOOP4:INDEX=INDEX-1;.IF.INDEX
    .GT.0.THEN..BGIN.LINK=LNKR.(CONT.(
    LINK)).;.GOTO.LOOP4.END..ELSE..BGIN.
    SETDIR.(-1,-1,LINK,OPRND(IK)).;.GOTO.CYCLE
    .END..END..ELSE..GOTO.ERROR;

.TAIL.
(list suffix)
    TAIL: TEMP=LNKR.(OPRND(IK)).;.IF.
    IOPRND(IK).NE.3.OR.TEMP.EQ.0.THEN.
    .GOTO.ERROR.ELSE.TEMP=LNKR.(CONT
    .(TEMP)).;.IF.TEMP.EQ.0.THEN..GOTO.
    ERROR.ELSE..BGIN.OPRND(IK)=TEMP;
    SETIND.(-1,1+LNKL.(CONT.(TEMP)).,-1,TEMP).;
    .GOTO.CYCLE.END.;

.CNCT.
(concatenation)
    CONCAT:.IF.IOPRND(IK).NE.3.OR.
    IOPRND(IK-1).NE.3.THEN..GOTO.
    ERROR.ELSE..BGIN.COPY.(LNKR.(OPRND(IK)).,
    JTOP,JBOT).;COPY.(LNKR.(OPRND(IK-1)).,
    ITOP,IBOT).;SETIND.(-1,-1,JTOP,IBOT).;
    IK=IK-1;OPRND(IK)=ITOP;.GOTO.CYCLE
    .END.

.LNGT.
(List length)
    LENGTH:.IF.IOPRND(IK).NE.3
    .THEN..GOTO.ERROR.ELSE.BGIN.COUNT=0;
    TEMP=LNKR.(OPRND(IK)).;IOPRND(IK)=0;
    LOOPS:.IF.TEMP.EQ.0.THEN..BGIN.OPRND(IK)
    =COUNT;.GOTO.CYCLE.END..ELSE..BGIN.
    COUNT=1+COUNT;TEMP=LNKR.(CONT
    .(TEMP)).;.GOTO.LOOPS.END.

```

### Reusage of Discarded List Cells by the System

The EULER programmer will not usually concern himself with the problem of "garbage collection" of discarded list cells in his programs, since garbage collection is automatic in EULER. This collection occurs at three points in EULER programs: at the .END. command of a block, at an assignment statement when the = command is executed, and between statements when the ; command is executed. At the end of a block, all lists local to the block are linked one after another to the last cell in the LAVS list. When an assignment statement assigns a new value to a variable that was formerly a list, that list is checked to see if its first cell only has one name. If that list only has one name, it is linked to the last cell of LAVS. When a semicolon is encountered in the translated program, the top of the OPRND stack is checked to see if it contains a link to some list having no name (such as would be left there if something like .OUT..( ). were the preceding statement). If there is a link to an anonymous list on the OPRND stack, then this link is removed from OPRND, and the list is joined to the end of LAVS.

As in the SLIP system (a part of which is used in our implementation of EULER), sublists of lists are joined to the end of LAVS only when the NUCELL routine encounters them in the process of fetching list cells from the top of LAVS. In this case, the sublists are only considered to be reusable if they are sublists of no more than one list (in this case, they are sublists of LAVS). More information on this garbage collection technique can be found in the original SLIP paper. (11)

### Communication of Data

The EULER language allows simple variables and lists to be printed out at execution time. A program statement such as

.OUT.V;

is translated into the sequence

.VRBL.V      .IN.    .OUT. ;

If V is a list, then the entire list, including sublists will be printed out. If V is a variable, then just that variable's value will be printed out. If V is a procedure, reference, or label, then PROCDR, REFRNC, or LABEL will be the appropriate "values" to be printed by the .OUT. operator.

As a further example, we might have the following statement in an EULER program:

```
.OUT. .(* N= ,3.2,.*THIS I,.*S A LI,.*ST WIT,
      .*H SUBL,.*ISTS. ,.($3$,.*AT.N,.(2,3).).);
```

When the corresponding statement in the translated program is executed, the following printed output will result:

```
(      N=          3.2000,THIS IS A LI
ST WITH SUBLISTS. ,   (  PROCDR,REFRNC,
(      2.0000,      3.0000   )      )
```

Note that commas are suppressed between list elements containing characters. A discussion of how one might extend the printing facilities of EULER can be found in the section entitled "Improvements and Extensions of EULER".

There are no facilities for reading data in the EULER language. The original EULER paper suggests that data be translated with the program, since translation can be a very rapid process. Perhaps a better suggestion, and one that we hope to implement, is to allow a list-structured input of data similar to the one used with the .OUT. command. The "Improvements and Extensions" Section of this paper has more to say on this matter.

The following algorithm implements list and variable printing as described above. The OUTPUT subroutine prints one datum for each call, and an index is used by the subroutine to position the datum on a printed line. Note that the LSTRTN stack facilitates writing out of sublists.

```
.OUT.      OUT: TYPE=IOPRND(IK);.IF.TYPE.NEQ.3
           .THEN..BGIN.INDEX=1; OUTPUT.(TYPE,OPRND(IK),
           .AT.INDEX);.GOTO.CYCLE.END..ELSE..IF.LNKR.(
           OPRND(IK))..EQ.0.THEN..BGIN.INDEX=1;
           OUTPUT.(7,.* (  ) ,.*AT.INDEX);.GOTO.CYCLE.END.
           .ELSE..BGIN.
           LSTCNT=INDEX=0;OUTPUT.(7,.* (  ,.*AT.INDEX).
           ;LINK=LNKR.(OPRND(IK));LOOP6:CTLNIK=CONT.(
           LINK);.IF.ID.(CTLNIK)..EQ.3.THEN..BGIN.
           LSTCNT=1+LSTCNT;LSTRTN(LSTCNT)=LINK;
           LINK=LNKR.(CONT.(LINK+1).);OUTPUT.(7,.* (  ,
           .AT.INDEX);.GOTO.LOOP6.END.
```

```

.ELSE..BGIN.OUTPUT.(ID.(CTLINK)..CONT.(LINK+1)..,
.AT.INDEX)..;.IF.ID.(CTLINK)..NEQ.7.AND.
LNKR.(CTLINK)..GT.O.THEN.OUTPUT.(7, , ,
.AT.INDEX)..ELSE..UNDF..END.;
.IF.LNKR.(CTLINK)..GT.O.THEN..BGIN.LINK=
LNKR.(CTLINK)..;.GOTO.LOOP6.END..ELSE.
.BGIN.LOOP7:OUTPUT.(7,. * ) ,.AT.INDEX)..;.IF.
LSTCNT.EQ.O.THEN..GOTO.CYCLE
.ELSE..BGIN.LINK=LNKR.(CONT.(LSTRN(LSTCNT))..).
;LSTCNT=LSTCNT-1;.IF.LINK.EQ.O.THEN.
.GOTO.LOOP7.ELSE..BGIN.OUTPUT.(7,. * , ,.AT.
INDEX)..;.GOTO.LOOP6.END..END.
.END.

```

### Acknowledgements

During the programming of the EULER reverse Polish interpreter, Charles Mesztenyi and John Pfaltz of the University of Maryland Computer Science Center very kindly offered suggestions on programming techniques that materially hastened the completion of the interpreter. The author also wishes to acknowledge discussions on the subject of the translator with John C. Reynolds of the Argonne National Laboratory and Martin Milgram of the University of Maryland. Daniel Fishman and Frank Adair of the Computer Center also gave of their time in the early programming of the interpreter.

## Bibliography

1. Floyd, R. W. A Descriptive Language for Symbol Manipulation.  
J. ACM 8 (Oct. 1961), 579-584.
2. Irons, E. T. A Syntax-Directed Compiler for ALGOL 60. C. ACM 4  
(Jan. 1961), 51-55.
3. Lewis, P. M. and Stearns, R. E. Syntax-Directed Transduction.  
J. ACM 15 (July 1968), 465-488.
4. Mills, David L. The Syntactic Structure of MAD/1. Defense  
Documentation Center Report Number AD-671-683, June, 1968.
5. Naur, P. (Ed.) Revised Report on the Algorithmic Language ALGOL 60.  
C. ACM 6 (Jan. 1963), 1-17.
6. Schneider, V. B. The Design of Processors for Context-Free Languages.  
National Science Foundation Memorandum, Department of Industrial  
Engineering and Management Science, Northwestern University,  
Evanston, Illinois, August, 1965.
7. \_\_\_\_\_. Pushdown-Store Processors of Context-Free Languages.  
Doctoral Dissertation, Northwestern University, Evanston,  
Illinois, 1966.
8. \_\_\_\_\_. Syntax-Checking and Parsing of Context-Free Languages  
by Pushdown-Store Automata. Proc. Spring Joint Comp. Conf.  
(1967), 71-75.
9. \_\_\_\_\_. A System for Designing Fast Programming Language  
Translators. Technical Report 68-76, Computer Science Center,  
The University of Maryland, College Park, Md., July, 1968.
10. Van Wijngaarden (Ed.). Draft Report on the Algorithmic Language  
ALGOL 68. Mathematisch Centrum Report MR-93, Amsterdam,  
March, 1968.
11. Weizenbaum, J. A Symmetric List Processor. C. ACM 6 (Sept. 1963).  
524.
12. Wirth, N. A Generalization of ALGOL. C. ACM 6 (Sept. 1963), 547-554.
13. Wirth, N. and Weber, H. EULER: A Generalization of ALGOL and Its  
Formal Definition: Parts I and II. C. ACM 3 (Jan.-Feb. 1966),  
13-25 and 89-99.
14. Naur, P. (Ed.). Revised Report on the Algorithmic Language  
ALGOL 60. C. ACM 6 (Jan. 1963), 1-17.



## Appendix 1. Features of the EULER Language

EULER is a nested block-structure language, similar to ALGOL. Thus, every block, consisting of a sequence of statements surrounded by .BGIN. and .END. parentheses, can be treated as a single statement in ALGOL fashion. An EULER program consists of an EULER block preceded by .ENTRY and followed by .EXIT..

In EULER, there are three declarations, one category for data variables, one for program labels, and one for formal parameters of procedures. In the program

```
".ENTRY .BGIN. .NEW.X,Y.,.LABL.Z.,
      Z..X=Y .END. .EXIT."
```

X and Y will store data, and Z will be a label preceding some statement.

Assigning a data type to a declared variable is accomplished by writing an assignment statement with data of the appropriate type on the right-hand side of the assignment. Thus, typing of variables in EULER is dynamic, since any assignment statement can change the data type stored in a variable. And, data typing is implicit, since there are no declarations like real, integer, etc., as appear in ALGOL. The following is a list of the eight EULER data types:

- I. Number - In the EULER system, all numbers are assumed to be floating point numbers. The assignment statement
 

```
"V=E.,"
```

 with E a numerical expression or number, causes variable V to become a numerical variable.
- II. Symbol - In this EULER implementation, an assignment statement such as
 

```
"V=.*ALPHAN.,"
```

 causes the six characters "ALPHAN" to be stored in the location named by variable V.
- III. Logical - The logical constants are .TRUE. and .FLSE., standing respectively for logical truth and falsehood. The assignment statement,
 

```
"V=L.,"
```

 with L a logical constant or logical expression, causes variable V to become a logical variable.

IV. Label - It will be recalled that EULER programs declare .NEW. variables as well as .LABL. statement labels. Interestingly enough, if V is a variable in some EULER program, and V is not in a block global to the block of label L, then the assignment statement

"V=L.,"

causes V henceforth to be of type label, and to be interchangeable with L in .GOTO. statements.

V. Reference - In EULER, if V1 is a variable not in a block global to the block of variable V2, then the assignment statement

"V1=.AT.V2.,"

makes V1 a pointer to the data stored in V2. After V1 is turned into such a pointer, the two statements

"V2=V2+1.,"

and "V1.IN.=V1.IN.+1.,"

will have exactly the same effect of manipulating whatever data is stored in V2.

VI. Procedure - An assignment statement of the form

"V1=.\$<expr>\$..,"

causes V1 to become the name of a parameterless procedure call with body given by <expr>. As a programming example, we might consider the following EULER block:

```
".BGIN..NEW.X,Y ., X=2;
      Y=.$FRML.Z., X=X+Z$..,
      .OUT.Y.(5)..END."
```

When Y.(5). is operated on by the .OUT. operator, the value 7.0000 will be written out.

VII. List - In EULER, lists can be constructed in three distinct ways:

(a) On command: "V1=.LIST.300.,"

This statement creates a list of 300 cells and makes V1 their name.

- (b) By explicit notation: "V2=.(1,.(2,3),4)..,"

This statement creates a list consisting of two numbers and a sublist and makes V2 the name of that list.

- (c) By concatenation: "V1=V1.CNCT.V2.,"

Using the concatenation operator, small lists can be joined together into larger ones. In addition, lists can be subscripted in the same way as ALGOL arrays, and each element of a list can be any EULER data type, including label, reference, and procedure. The following EULER block is an example of the generality of the list notation:

```
.BGIN..NEW.X,Y.,.LABL.Z.,
  Y=.(2,.$.BGIN.X=X+1.,Y(X).END.$.,
    $.OUT.X$,Z)..,
  X=Y(1),Y.(X)..,GOTO.Y(4)..,
  Z...OUT.*FINISH.END.
```

With this program segment, first 3.0000, then FINISH, will be written out by the executed program.

VIII. Undefined - Every variable declared by .NEW. in an EULER program is initially of type .UNDF.(undefined). In addition .UNDF. is used as a data constant occasionally and as an empty option in conditional statements, such as:

```
"V=.IF.L1.THEN..(1,5)..ELSE..UNDF..,"
```

For more details on EULER programming the reader is referred to the Wirth and Weber EULER paper (11).

## Appendix 2. The SLIP Routines Used in Implementing EULER

The current EULER implementation uses lists whose cells consist of pairs of memory words having the following format:

|                         |       |      |      |
|-------------------------|-------|------|------|
| Machine Address = M     | Type  | LNKL | LNKR |
| Machine Address = M + 1 | Datum |      |      |

The LNK R field of word M stores the machine address of the first word of the next cell in a list. If LNK R = 0, then that cell is the last cell of some list. The LNK L field stores a count of how many program names refer to a cell; i.e., the LNK L measures how many times a programmer has made that cell the head of a list. The type field, of course, stores the type code of the datum or link contained in word M + 1.

In the EULER algorithms describing the commands of the reverse Polish interpreter system, the following subroutines are used for list manipulation:

I. INITAS.(.AT.AVSL,.AT.LAVS,2000).

This routine passes pointers from the word AVSL and the 200 word array LAVS to a subroutine that organizes LAVS into a pool of list cells. A pointer to the first cell of this pool is stored in the LNK R field of AVSL, and the LNK L field of AVSL stores a pointer to the last cell of the LAVS pool.

II. ID.(WORD)., LNK L.(WORD)., LNK R.(WORD).

These functions perform appropriate masking and shifting of the contents of the <name> WORD, and return as values the ID, LNK L, and LNK R fields of WORD, respectively.

III. CONT.(ADDRESS).

This function returns the datum stored in the memory word with location given by ADDRESS.

IV. SETDIR.(ID,LNKL,LNKR,NAME).

Packs the value of ID into the ID field, LNKL into the LNKL field, and LNKR into the LNKR field of the word NAME. If any of the first three parameters of SETDIR is -1, the corresponding field of NAME is left unchanged.

V. SETIND.(ID,LNKL,LNKR,ADDRESS).

Similar to SETDIR above, except that ADDRESS is the machine address of some word in memory.

VI. MADOV,(NAME).

This function returns the machine address of the word containing the data stored in NAME.

VII. NUCELL

This parameterless function returns a link to the first free list cell in the LAVS pool. NUCELL also resets the LNKR field of the AVSL word to point to the new head of LAVS, and NUCELL initiates garbage collection of any unused sublists that it encounters.

```

$EXECUTE      IBJOB
$ID           SCHNEIDER,VICTOR B.*001/68/799*5M*50P*100C*BINARY$
$IBJOB       GO
$IBFTC TRANSL

C NAMING AND TYPING OF STRUCTURES IN THE EULER TRANSLATOR.
  LOGICAL UNSAME,NOTIDT
  DIMENSION IDNTLS(1000),INPUT(3000),OUTPUT(1000),N(300),
1    STORAJ(200)
  EQUIVALENCE (INPUT(501),OUTPUT(1)),(INPUT(1501),IDNTLS(1)),
1    (INPUT(2501),N(1)), (INPUT(2801),STORAJ(1))
  INTEGER OUTPUT,PROCHD,PRCEND,BEGIN,TRMNAT,SEMCLN,COLON,ASSINE,
1    OUT,TEST,SWITCH,THEN,ELSE,OR, FRSTOR,SCNDOR,AND,FSTAND,SCNAND,
2    GREATR,GRATEQ,PLUS,ADD,SUBTRC,FLTDIV,EXPNT,CONCAT,UNDFND,
3    RTPREN,TAIL,FORMAL,COMMA,VARIBL,AT,TOREAL,TOABSL,TOINTG,TRUTH,
4    FALSTY,SYMBOL,BLKNUM,STORAJ
  DATA KBGIN,KEND/6H.ENTRY,6H.EXIT./
  DATA IBLANK,NUMBER,LENGTH,LOGCAL/
1    1H0,6H.NMBR.,6H.LNGT.,6H.LGCL./
  DATA ISNMBR,ISLGCL,ISLABL,ISLIST,ISUNDF,ISRFRN,ISPRCD,ISYMBL/
1    6H.ISNU.,6H.ISLO.,6H.ISLA.,6H.ISLI.,6HCISUN.,6H.ISRE.,
2    6H.ISPR.,6H.ISSY./
  DATA NEW,LABEL,JUMP,NEGATE,LABDEF,ISSAME,NOTSAM/
1    5H.NEW.,6H.LABL.,6H.GOTO.,5H.NOT.,6H.LBDF.,4H.EQ.,5H.NEQ./
  DATA LESSTN,LESSEQ,NEG,MODULO,MLTPLY,INTDIV/
1    4H.LT.,5H.LEQ.,5H.NEG.,6H.MDLO.,1H*, 3H././
  DATA INSIDE,LFPREN,LSCALL,LIST,LISTHD,LSTEND,LSBSCP/
1    4H.IN.,1H(,6H.LSCL.,6H.LIST.,2H.(,2H),,4H(.,./
  DATA PROCHD,PRCEND,BEGIN,TRMNAT,SEMCLN,COLON/
1    2H.$,2H$,.,6H.BGIN.,5H.END.,2H.,,2H../
  DATA ASSINE,OUT,TEST,SWITCH,THEN/
1    1H=,5H.OUT.,4H.IF.,6H.SWCH.,6H.THEN./
  DATA GREATR,GRATEQ,PLUS,ADD,SUBTRC,FLTDIV,EXPNT/
1    4H.GT.,5H.GEQ.,6H.PLUS.,1H+,1H-,1H/,2H**/
  DATA ELSE,OR,FRSTOR,SCNDOR,AND,FSTAND,SCNAND/
1    6H.ELSE.,4H.OR.,6H.FSOR.,6H.SCOR.,5H.AND.,6H.FAND.,6H.SAND./
  DATA CONCAT,UNDFND,RTPREN,TAIL,FORMAL,COMMA,VARIBL/
1    6H.CNCT.,6H.UNDF.,1H),6H.TAIL.,6H.FRML.,1H,6H.VRBL./
  DATA AT,TOREAL,TOABSL,TOINTG,TRUTH,FALSTY,SYMBOL/
1    4H.AT.,6H.REAL.,6H.ABST.,6H.INTG.,6H.TRUE.,6H.FALSE.,2H.*//

C
C INITIALIZATION--IT INDEXES IDNTLS, JK INDEXES INPUT, LL INDEXES OUTPUT
C AND IK INDEXES N, BLKNUM INDEXES STORAJ. MARK IS USED IN WRITING OUT
C TRACES FOR BAD PROGRAMS.
C
  IT=0
  JK=1
  LL=0
  IK=0
  BLKNUM=0
  MARK=1
  CALL EDITOR(INPUT)
  WRITE(6,4)
4    FORMAT(1H1)

C
C INITIAL POINT FOR TRANSLATION
C
  IF(UNSAME(INPUT(JK),KBGIN)) GO TO 8000
  CALL STORE(KBGIN,N,IK,JK)

C
C INITIAL POINT FOR BLOCK.
C

```

```

100      ICODE=INPUT(JK)
        IF(UNSAFE(ICODE,ISNMBR).AND.UNSAFE(ICODE,ISLGCL).AND.
1          UNSAFE(ICODE,ISLABL).AND.UNSAFE(ICODE,ISLIST).AND.
2          UNSAFE(ICODE,ISUNDF).AND.UNSAFE(ICODE,ISRFRN).AND.
3          UNSAFE(ICODE,ISPRCD).AND.UNSAFE(ICODE,ISYMBL)) GO TO 110
        CALL STORE(ICODE,N,IK,JK)
        GO TO 200
C      200 IS INITIAL POINT OF PRIMARY.
110      IF(UNSAFE(ICODE,LISTHD)) GO TO 120
        CALL OUTCOD(LISTHD,OUTPUT,LL)
        CALL STORE(LISTHD,N,IK,JK)
        GO TO 100
120      IF(UNSAFE(ICODE,BEGIN).AND.UNSAFE(ICODE,PROCHD)) GO TO 130
        BLKNUM=BLKNUM+1
        STORAJ(BLKNUM)=IT+1
        CALL OUTCOD(ICODE,OUTPUT,LL)
        CALL STORE(ICODE,N,IK,JK)
        GO TO 100
130      IF(UNSAFE(ICODE,ADD)) GO TO 140
        CALL STORE(PLUS,N,IK,JK)
        GO TO 100
140      IF(UNSAFE(ICODE,SUBTRC)) GO TO 150
        CALL STORE(NEG,N,IK,JK)
        GO TO 100
150      IF(UNSAFE(ICODE,TAIL).AND.UNSAFE(ICODE,TOUT).AND.UNSAFE(ICODE,
1          NOT).AND.UNSAFE(ICODE,TEST).AND.UNSAFE(ICODE,THEN).AND.
2          UNSAFE(ICODE,ELSE).AND.UNSAFE(ICODE,LIST).AND.UNSAFE(ICODE,
3          LOGCAL).AND.UNSAFE(ICODE,TOREAL).AND.UNSAFE(ICODE,LENGTH)
4          .AND.UNSAFE(ICODE,AT).AND.UNSAFE(ICODE,TOINTG).AND.UNSAFE(
5          ICODE,TOABSL).AND.UNSAFE(ICODE,LIST).AND.UNSAFE(ICODE,JUMP))
6          GO TO 160
        CALL STORE(ICODE,N,IK,JK)
        GO TO 100
160      IF(UNSAFE(ICODE,NEW).AND.UNSAFE(ICODE,FORMAL).AND.
1          UNSAFE(ICODE,LABEL)) GO TO 200
        CALL STORE(ICODE,N,IK,JK)
C
C      INITIAL POINT FOR PRIMARY.
C
200      ICODE=INPUT(JK)
        IF(UNSAFE(ICODE,SYMBOL)) GO TO 210
        CALL OUTCOD(SYMBOL,OUTPUT,LL)
        JK=JK+1
        CALL OUTCOD(INPUT(JK),OUTPUT,LL)
        JK=JK+1
        GO TO 300
210      IF(UNSAFE(ICODE,NUMBER)) GO TO 220
        CALL OUTCOD(NUMBER,OUTPUT,LL)
        CALL OUTCOD(INPUT(JK+1),OUTPUT,LL)
        CALL OUTCOD(INPUT(JK+2),OUTPUT,LL)
        JK=JK+3
        GO TO 300
220      IF(UNSAFE(ICODE,TRUTH).AND.UNSAFE(ICODE,FALSTY).AND.UNSAFE(
1          ICODE,UNDFND)) GO TO 230
        CALL OUTCOD(ICODE,OUTPUT,LL)
        JK=JK+1
        GO TO 300
230      IF(NOTIDT(ICODE)) GO TO 8230
        IF(UNSAFE(INPUT(JK+1),COLON)) GO TO 235
        CALL OUTCOD(LABDEF,OUTPUT,LL)
        CALL OUTCOD(ICODE,OUTPUT,LL)

```

```

                JK=JK+2
                GO TO 100
235             JK=JK+1
                NOFIK=N(IK)
                IF(UNSAFE(NOFIK,FORMAL)) GO TO 250
                ITEMP=FORMAL
                CALL OUTCOD(FORMAL,OUTPUT,LL)
                CALL OUTCOD(ICODE,OUTPUT,LL)
                IT=IT+1
                IDNTLS(IT)=ICODE
                IK=IK-1
                ICODE=INPUT(JK)
                IF(UNSAFE(ICODE,COMMA)) GO TO 240
                CALL STORE(ITEMP,N,IK,JK)
                GO TO 200
240             IF(UNSAFE(ICODE,SEMCLN)) GO TO 8240
                JK=JK+1
                IF(UNSAFE(N(IK),PROCHD)) GO TO 8241
                GO TO 100
250             IF(UNSAFE(NOFIK,NEW).AND.UNSAFE(NOFIK,LABEL)) GO TO 260
                ITEMP=NOFIK
                CALL OUTCOD(ITEMP,OUTPUT,LL)
                CALL OUTCOD(ICODE,OUTPUT,LL)
                IT=IT+1
                IDNTLS(IT)=ICODE
                IK=IK-1
                ICODE=INPUT(JK)
                IF(UNSAFE(ICODE,COMMA)) GO TO 255
                CALL STORE(ITEMP,N,IK,JK)
                GO TO 200
255             IF(UNSAFE(ICODE,SEMCLN)) GO TO 8255
                JK=JK+1
                IF(UNSAFE(N(IK),BEGIN)) GO TO 8256
                GO TO 100
C             ICODE IS STILL THE IDENTIFIER IN QUESTION.
260             ITEMP=IT
261             IF(.NOT.UNSAFE(IDNTLS(ITEMP),ICODE)) GO TO 265
                ITEMP=ITEMP-1
                IF(ITEMP) 8261,8261,261
265             CALL OUTCOD(VARIBL,OUTPUT,LL)
                CALL OUTCOD(ICODE,OUTPUT,LL)
C
C             266 IS THE LABEL VARIABLE IN THE FLOWCHARTC
C             JK WAS INCREMENTED AT LABEL 235.ICODE WAS THEN INPUT(JK-1).
C
266             ICODE=INPUT(JK)
                IF(UNSAFE(ICODE,INSIDE)) GO TO 269
                CALL OUTCOD(INSIDE,OUTPUT,LL)
                JK=JK+1
                GO TO 266
269             IF(UNSAFE(ICODE,ASSINE)) GO TO 275
270             CALL STORE(ICODE,N,IK,JK)
                GO TO 100
275             IF(UNSAFE(ICODE,LISTHD)) GO TO 277
                CALL OUTCOD(LSCALL,OUTPUT,LL)
                CALL OUTCOD(LISTHD,OUTPUT,LL)
                GO TO 270
277             IF(UNSAFE(ICODE,LFPREN)) GO TO 280
                CALL STORE(LSBSCP,N,IK,JK)
                GO TO 100
280             NOFIK=N(IK)

```



```

      IF(UNSAFE(NOFIK,AT)) GO TO 285
      IK=IK-1
      GO TO 300
285   IF(UNSAFE(NOFIK,ISNMBR).AND.UNSAFE(NOFIK,ISLGCL).AND.
      1   UNSAFE(NOFIK,ISLABL).AND.UNSAFE(NOFIK,ISLIST).AND.
      2   UNSAFE(NOFIK,ISUNDF).AND.UNSAFE(NOFIK,ISRFRN).AND.
      3   UNSAFE(NOFIK,ISPRCD).AND.UNSAFE(NOFIK,ISYMBL)) GO TO 295
      CALL OUTCOD(NOFIK,OUTPUT,LL)
      IK=IK-1
      GO TO 300
295   CALL OUTCOD(INSIDE,OUTPUT,LL)
C
C   300 IS THE LABEL OF PRIMARY.
C
300   NOFIK=N(IK)
      IF(UNSAFE(NOFIK,TAIL)) GO TO 305
      CALL OUTCOD(NOFIK,OUTPUT,LL)
      IK=IK-1
      GO TO 300
305   IF(UNSAFE(NOFIK,JUMP).AND.UNSAFE(NOFIK,OUT)) GO TO 310
      CALL OUTCOD(NOFIK,OUTPUT,LL)
      IK=IK-1
      GO TO 1000
C   1000 IS THE LABEL OF EXPRESSION.
310   IF(UNSAFE(NOFIK,CONCAT)) GO TO 400
C   400 IS THE LABEL OF CATENA.
      CALL OUTCOD(NOFIK,OUTPUT,LL)
      IK=IK-1
C
C   400 IS THE LABEL OF CATENA.
C
400   IF(UNSAFE(INPUT(JK),CONCAT)) GO TO 410
      CALL STORE(CONCAT,N,IK,JK)
      GO TO 100
410   NOFIK=N(IK)
      IF(UNSAFE(NOFIK,LENGTH)) GO TO 420
      CALL OUTCOD(LENGTH,OUTPUT,LL)
      IK=IK-1
      GO TO 300
420   IF(UNSAFE(NOFIK,EXPNT)) GO TO 500
      CALL OUTCOD(EXPNT,OUTPUT,LL)
      IK=IK-1
C
C   500 IS THE LABEL OF FACTOR.
C
500   IF(UNSAFE(INPUT(JK),EXPNT)) GO TO 520
      CALL STORE(EXPNT,N,IK,JK)
      GO TO 100
520   NOFIK=N(IK)
      IF(UNSAFE(NOFIK,MLTPLY).AND.UNSAFE(NOFIK,INTDIV).AND.
      1   UNSAFE(NOFIK,FLTDIV).AND.UNSAFE(NOFIK,MODULO)) GO TO 600
      CALL OUTCOD(NOFIK,OUTPUT,LL)
      IK=IK-1
C
C   600 IS THE LABEL OF TERM.
C
600   ICODE=INPUT(JK)
      IF(UNSAFE(ICODE,MLTPLY).AND.UNSAFE(ICODE,INTDIV).AND.
      1   UNSAFE(ICODE,FLTDIV).AND.UNSAFE(ICODE,MODULO)) GO TO 610
      CALL STORE(ICODE,N,IK,JK)
      GO TO 100

```

```

610      NOFIK=N(IK)
        IF(UNSAFE(NOFIK,PLUS).AND.UNSAFE(NOFIK,NEG).AND.UNSAFE(NOFIK,
1         ADD).AND.UNSAFE(NOFIK,SUBIRC)) GO TO 700
        CALL OUTCOD(NOFIK,OUTPUT,LL)
        IK=IK-1
C
C      700 IS THE LABEL OF SUM.
C
700      ICODE=INPUT(JK)
        IF(UNSAFE(ICODE,ISSAME).AND.UNSAFE(ICODE,NOTSAM).AND.
1         UNSAFE(ICODE,LESSTN).AND.UNSAFE(ICODE,LESSEQ).AND.
2         UNSAFE(ICODE,GREATR).AND.UNSAFE(ICODE,GRATEQ).AND.
3         UNSAFE(ICODE,ADD).AND.UNSAFE(ICODE,SUBIRC)) GO TO 710
        CALL STORE(ICODE,N,IK,JK)
        GO TO 100
710      NOFIK=N(IK)
        IF(UNSAFE(NOFIK,TOINIG).AND.UNSAFE(NOFIK,IOABSL).AND.
1         UNSAFE(NOFIK,LOGCAL).AND.UNSAFE(NOFIK,LIST)) GO TO 720
        CALL OUTCOD(NOFIK,OUTPUT,LL)
        IK=IK-1
        GO TO 300
720      IF(UNSAFE(NOFIK,ISSAME).AND.UNSAFE(NOFIK,NOTSAM).AND.
1         UNSAFE(NOFIK,LESSTN).AND.UNSAFE(NOFIK,LESSEQ).AND.
2         UNSAFE(NOFIK,GREATR).AND.UNSAFE(NOFIK,GRATEQ)) GO TO 730
        CALL OUTCOD(NOFIK,OUTPUT,LL)
        IK=IK-1
        GO TO 800
730      IF(UNSAFE(NOFIK,LSBSCP)) GO TO 800
        IK=IK-1
        IF(UNSAFE(INPUT(JK),RTPREN)) GO TO 8730
        CALL OUTCOD(RTPREN,OUTPUT,LL)
        JK=JK+1
        GO TO 266
C
C      800 IS THE LABEL OF RELATION.
C
800      IF(UNSAFE(N(IK),NEGATE)) GO TO 850
        CALL OUTCOD(NEGATE,OUTPUT,LL)
        IK=IK-1
C
C      850 IS NEGATION.
C
850      IF(UNSAFE(N(IK),AND)) GO TO 870
        CALL OUTCOD(SCNAND,OUTPUT,LL)
        IK=IK-1
C
C      870 IS THE LABEL OF DISJUNCTION
C
870      IF(UNSAFE(INPUT(JK),AND)) GO TO 880
        CALL STORE(AND,N,IK,JK)
        CALL OUTCOD(FSTAND,OUTPUT,LL)
        GO TO 100
880      IF(UNSAFE(N(IK),OR)) GO TO 900
        CALL OUTCOD(SCNDOR,OUTPUT,LL)
        IK=IK-1
C
C      900 IS THE LABEL OF CONJUNCTION.
C
900      IF(UNSAFE(INPUT(JK),OR)) GO TO 910
        CALL STORE(OR,N,IK,JK)
        CALL OUTCOD(FRSTOR,OUTPUT,LL)

```

```

GO TO 100
910  IF(UNSAFE(N(IK),TOREAL)) GO TO 1000
    CALL OUTCOD(TOREAL,OUTPUT,LL)
    IK=IK-1
    GO TO 300
C 1000 IS THE LABEL OF EXPRESSION.
1000 NOFIK=N(IK)
    IF(UNSAFE(NOFIK,ASSINE)) GO TO 1010
    CALL OUTCOD(ASSINE,OUTPUT,LL)
    IK=IK-1
    GO TO 1000
1010 IF(UNSAFE(NOFIK,LFPREN)) GO TO 1020
    IK=IK-1
    IF(UNSAFE(INPUT(JK),RTPREN)) GO TO 9010
    JK=JK+1
    GO TO 300
1020 IF(UNSAFE(NOFIK,TEST)) GO TO 1030
    CALL OUTCOD(TEST,OUTPUT,LL)
    IK=IK-1
    IF(UNSAFE(INPUT(JK),THEN)) GO TO 9020
    CALL STORE(THEN,N,IK,JK)
    GO TO 100
1030 IF(UNSAFE(NOFIK,THEN)) GO TO 1040
    CALL OUTCOD(SWITCH,OUTPUT,LL)
    CALL OUTCOD(THEN,OUTPUT,LL)
    CALL OUTCOD(NEGATE,OUTPUT,LL)
    CALL OUTCOD(TEST,OUTPUT,LL)
    IF(UNSAFE(INPUT(JK),ELSE)) GO TO 9030
    N(IK)=ELSE
    JK=JK+1
    GO TO 100
1040 IF(UNSAFE(NOFIK,ELSE)) GO TO 1050
    CALL OUTCOD(SWITCH,OUTPUT,LL)
    CALL OUTCOD(THEN,OUTPUT,LL)
    CALL OUTCOD(SEMCLN,OUTPUT,LL)
    IK=IK-1
    GO TO 1000
1050 IF(UNSAFE(NOFIK,LISTHD)) GO TO 1060
    IK=IK-1
    IF(UNSAFE(INPUT(JK),LSTEND)) GO TO 1055
    CALL OUTCOD(LSTEND,OUTPUT,LL)
    JK=JK+1
    GO TO 300
1055 IF(UNSAFE(INPUT(JK),COMMA)) GO TO 9055
    CALL OUTCOD(COMMA,OUTPUT,LL)
    CALL STORE(LISTHD,N,IK,JK)
    GO TO 100
1060 IF(UNSAFE(NOFIK,PROCHD)) GO TO 1100
    IK=IK-1
    IF(UNSAFE(INPUT(JK),PRCEND)) GO TO 9060
    JK=JK+1
    CALL OUTCOD(PRCEND,OUTPUT,LL)
    GO TO 300
C
C 1100 IS THE LABEL OF STATEMENT.
C
1100 IF(UNSAFE(N(IK),SEMCLN)) GO TO 1110
    IK=IK-1
1110 ICODE=INPUT(JK)
    IF(UNSAFE(ICODE,SEMCLN)) GO TO 1120
    CALL OUTCOD(SEMCLN,OUTPUT,LL)

```

```

        CALL STORE(SEMCLN,N,IK,JK)
        GO TO 100
1120    IF(UNSAFE(ICODE,TRMNAT)) GO TO 9120
1121    CALL OUTCOD(TRMNAT,OUTPUT,LL)
        JK=JK+1
        IF(UNSAFE(N(IK),BEGIN)) GO TO 9121
        IT=STORAJ(BLKNUM)-1
        BLKNUM=BLKNUM-1
        IK=IK-1
        IF(IT.LT.0) GO TO 9121
C
C      1200 IS THE LABEL OF BLOCK.
C
1200    IF(UNSAFE(N(IK),KBGIN)) GO TO 1000
        IF(UNSAFE(INPUT(JK),KEND)) GO TO 9200
        IIK=LL+1
        DO 1210 IKK=IIK,1000
1210    OUTPUT(IKK)=IBLANK
        IIK=0
1220    IIK=IIK+1
        IKK=11*IIK
        ITT=IKK-10
        IF(IKK.GT.1000) IKK=1000
        IF(IKK.EQ.1000.OR.OUTPUT(IKK+1).EQ.IBLANK) GO TO 1240
1230    WRITE(7,1235)(OUTPUT(NOFIK),NOFIK=ITT,IKK)
        GO TO 1220
1240    WRITE(7,1245)MLTPLY,(OUTPUT(NOFIK),NOFIK=ITT,IKK)
1235    FORMAT(6X,11A6,8X)
1245    FORMAT(A1,5X,11A6,8X)
        STOP
6000    WRITE(6,6080)
6080    FORMAT(1H0,82HTHE FOLLOWING IS A TRACE OF THE PORTION OF YOUR
1PROGRAM THAT CONTAINED THE ERROR.          ,/,56HTHE LAST WORD CAUSED
2THIS ERROR MESSAGE TO BE WRITTEN.          )
6020    WRITE(6,6050)(INPUT(MM),MM=MARK,JK)
6050    FORMAT(1H ,12A6)
        WRITE(6,6060)
6060    FORMAT(1H0,25HTHE ERROR SCAN CONTINUES./////////)
6090    JK=JK+1
        IF(JK.GT.500) GO TO 9500
        ICODE=INPUT(JK)
        IF(UNSAFE(ICODE,SEMCLN)) GO TO 6095
        JK=JK+1
        MARK=JK
        GO TO 100
6095    IF(UNSAFE(ICODE,TRMNAT)) GO TO 6090
        MARK=JK
        JK=JK+1
        GO TO 1110
8000    WRITE(6,4000)
4000    FORMAT(1H0,50HIMPROPER CONTROL SYMBOL AT BEGINNING OF PROGRAM.
1      )
        JK=JK+1
        GO TO 100
8230    WRITE(6,4230)
4230    FORMAT(1H0,50HILLEGAL OR MALFORMED IDENTIFIER. PROGRAM INVALID
1.    )
        GO TO 6000
8240    WRITE(6,4240)
4240    FORMAT(1H0,60HILLEGAL PUNCTUATION IN FORMAL DECLARATION. INVAL
1ID PROGRAM.    )

```

```

      GO TO 6000
8241  WRITE(6,4241)
4241  FORMAT(1H0,60HFORMAL DECLARATION IN NONPROCEDURE BLOCK. INVALID
      1D PROGRAM.      )
      GO TO 6000
8255  WRITE(6,4255)
4255  FORMAT(1H0,70HILLEGAL PUNCTUATION IN DECLARATION PART OF BLOCK
      1. INVALID PROGRAM.      )
      GO TO 6000
8256  WRITE(6,4256)
4256  FORMAT(1H0,67HDECLARATION ATTEMPT AT WRONG POINT IN PROGRAM.
      1INVALID PROGRAM.      )
      GO TO 6000
8261  WRITE(6,4261)
4261  FORMAT(1H0,60HATTEMPTED USE OF UNDECLARED IDENTIFIER. INVALID
      1PROGRAM.      )
      GO TO 6000
8730  WRITE(6,4730)
4730  FORMAT(1H0,60HUNBALANCED SUBSCRIPTING PARENTHESES. INVALID
      1PROGRAM.      )
      GO TO 6000
9010  WRITE(6,5010)
5010  FORMAT(1H0,67HUNBALANCED PARENTHESES AROUND AN EXPRESSION. INV
      1ALID PROGRAM.      )
      GO TO 6000
9020  WRITE(6,5020)
5020  FORMAT(1H0,60HMISSING THEN** IN CONDITIONAL STATEMENT. INVALID
      1 PROGRAM.      )
      GO TO 6000
9030  WRITE(6,5030)
5030  FORMAT(1H0,60HMISSING *ELSE* IN CONDITIONAL STATEMENT. INVALID
      1PROGRAM.      )
      GO TO 6000
9055  WRITE(6,5055)
5055  FORMAT(1H0,41HMISSING *COMMA* IN LIST. INVALID PROGRAM.      )
      GO TO 6000
9060  WRITE(6,5060)
5060  FORMAT(1H0,60HMISSING $. TO END A PROCEDURE DEFINITION. INVALID
      1D PROGRAM.      )
      GO TO 6000
9120  WRITE(6,5120)
5120  FORMAT(1H0,70HNO SEMICOLON** OR *END* AT END OF A STATEMENT.
      1INVALID PROGRAM.      )
      GO TO 6000
9121  WRITE(6,5121)
5121  FORMAT(1H0,60H*BEGINS* AND *ENDS* DO NOT BALANCE. INVALID
      1PROGRAM.      )
      GO TO 6000
9200  WRITE(6,5200)
5200  FORMAT(1H0,67HIMPROPER CONTROL CARD SYMBOLS. *$.* IS MISSING
      1AT END OF PROGRAM.      )
      GO TO 6000
9500  WRITE(6,6050)(OUTPUT(IKK),IKK=1,500)
      STOP
      END
$IBFT: STOREX
      SUBROUTINE STORE(ITEM,INPLAC,INDEXI,INDEXJ)
      DIMENSION INPLAC(300)
      INDEXI=INDEXI+1
      INPLAC(INDEXI)=ITEM
      INDEXJ=INDEXJ+1

```

```

        RETURN
        END
$IBFTC OUTCDX
        SUBROUTINE OUTCOD(ITEM,INPLAC,INDEXL)
            DIMENSION INPLAC(1000)
            INDEXL=INDEXL+1
            INPLAC(INDEXL)=ITEM
            IF(INDEXL+1.GT.1000) GO TO 5
            RETURN
5       WRITE(6,10)(INPLAC(INDEXL),INDEXL=1,1000)
10      FORMAT(1H0,5X,12A6)
        WRITE(6,11)
11      FORMAT(1H0,5X,31HPROGRAM TOO LARGE TO TRANSLATE.)
        STOP
        END
$IBFTC NOTIDX
        LOGICAL FUNCTION NOTIDT(NAME)
            LOGICAL LETTER
            IF(.NOT.LETTER(ISHIFT(FSTSYM(NAME)))) GO TO 5
            NOTIDT=.FALSE.
            RETURN
5       NOTIDT=.TRUE.
            RETURN
            END
$IBFT: EDITX
        SUBROUTINE EDITOR(INPUT)
C       INITIALIZATION
            LOGICAL DIGIT,LETTER
            INTEGER STRGHD
            DATA STRGHD/2H.*/
            INTEGER SEMCLN,PROCHD,PRIOD5,SPACE,BLANK,PERIOD,ASTRSK
            DATA ICOMMA,NUMBER/2H0,,6H.NMBR./
            DATA SEMCLN,PROCHD,LISTHD,PRIOD5/2H.,,2H$.2H.(,2H0./
            INTEGER ZERO,ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT
            DIMENSION INPUT(3000)
            DATA ZERO,ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT,NINE/
11H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9/
            DATA IBLANK,ASTRSK/2H0,1H*/
            DATA BLANK,PERIOD,IPLUS,IMINUS,IASIGN,IFLDIV,IDOLAR,
11IRPREN,IASTRK/1H,1H.,2H0+,2H0-,2H0=,2H0/,2H0$,2H0),2H0*/
            DATA ILPREN,IEXPNT/2H0(,2H**/
            DO 10 KK=1,46
                LM=66*KK
                LL=LM-65
                IF(LM.GT.3000) LM=3000
                READ(5,11)MARKER,(INPUT(KL),KL=LL,LM)
                IF(ISHIFT(MARKER).EQ.IASTRK) GO TO 13
10      CONTINUE
11      FORMAT(A1,5X,66A1,8X)
12      FORMAT(1H,80A1)
13      KK=0
            WRITE(6,12)(INPUT(KL),KL=1,LM)
            KL=0
15      KK=KK+1
            IF(KK.GT.3000)GO TO 80
            ICODE=ISHIFT(INPUT(KK))
            IF(ICODE.EQ.IBLANK)GO TO 15
            IF(ICODE.EQ.PRIOD5)GO TO 20
            IF(ICODE.EQ.IPLUS.OR.ICODE.EQ.IMINUS.OR.ICODE.EQ.IASIGN.OR.
1      ICODE.EQ.ICOMMA.OR.ICODE.EQ.IFLDIV.OR.ICODE.EQ.ILPREN)GO TO 25
            IF(ICODE.EQ.IDOLAR.OR.ICODE.EQ.IRPREN)GO TO 30

```

```

      IF(ICODE.EQ.IASTRK)GO TO 35
      IF(DIGIT(INPUT(KK)))GO TO 45
      IF(LETTER(ICODE))GO TO 40
      GO TO 15
C   RESERVED WORK BEGINNING WITH PERIOD
20   KL=KL+1
      KK=KK+1
      ICODE=ISHIFT(INPUT(KK))
      IF(ICODE.NE.          ICOMMA)GO TO 21
      INPUT(KL)=SEMCLN
      GO TO 15
21   IF(ICODE.NE.          IDOLAR)GO TO 22
      INPUT(KL)=PROCHD
      GO TO 15
22   IF(ICODE.NE.          ILPREN)GO TO 23
      INPUT(KL)=LISTHD
      GO TO 15
23   IF(ICODE.EQ.IASTRK) GO TO 2( )
      INPUT(KL)= SPACE(0,PERIOD)
      LL=1
24   INPUT(KL)=IPACK(INPUT(KL),      SPACE(6MLL,INPUT(KK)))
      KK=KK+1
      ICODE=ISHIFT(INPUT(KK))
      IF(ICODE.EQ.IBLANK)GO TO 4
      LL=LL+1
      IF(ICODE.EQ.PRIOD5)GO TO 3
      IF(LL.LT.6) GO TO 24
      KK=KK-1
      GO TO 15
3   INPUT(KL)=IPACK(INPUT(KL),      SPACE(6*LL,PERIOD))
4   LL=LL+1
      DO 5 LLL=LL,5
5   INPUT(KL)=IPACK(INPUT(KL),      SPACE(6*LLL,BLANK))
      GO TO 15
C   SINGLE CHARACTER OPERATOR
25   KL=KL+1
      INPUT(KL)=INPUT(KK)
      GO TO 15
C   FIRST CHARACTER OF OPERATOR WITH PERIOD AS SECOND CHARACTER
30   KL=KL+1
      IF(ISHIFT(INPUT(KK+1)).NE.PRIOD5)GO TO 31
      INPUT(KL)=IPACK(SPACE(0,INPUT(KK)),PRIOD5)
      KK=KK+1
      GO TO 15
31   INPUT(KL)=INPUT(KK)
      GO TO 15
C   FIRST ASTERISK OF POSSIBLE ASTERISK PAIR
35   KL=KL+1
      IF(ISHIFT(INPUT(KK+1)).NE.IASTRK)GO TO 36
      KK=KK+1
      INPUT(KL)=IEXPNT
      GO TO 15
36   INPUT(KL)=ASTRSK
      GO TO 15
C   FIRST LETTER OF AN IDENTIFIER
40   KL=KL+1
      INPUT(KL)=0
      LL=0
41   INPUT(KL)=IPACK(INPUT(KL),      SPACE(6*LL,INPUT(KK)))
      KK=KK+1
      LL=LL+1

```

```

        IF(LL.GT.5) GO TO 43
        IF(DIGIT(INPUT(KK)).OR.LETTER(ISHIFT(INPUT(KK))))GO TO 41
        DO 42 LLL=LL,5
42      INPUT(KL)=IPACK(INPUT(KL),      SPACE(6*LLL,BLANK))
43      KK=KK-1
        GO TO 15
C      FIRST DIGIT OF NUMBER
45      KL=KL+1
        INPUT(KL)=NUMBER
        KL=KL+1
        TEMP=0.0
46      IF(INPUT(KK).NE.ZERO) GO TO 47
        ADDEND=0.0
        GO TO 56
47      IF(INPUT(KK).NE.ONE) GO TO 48
        ADDEND =1.0
        GO TO 56
48      IF(INPUT(KK) .NE.TWO) GO TO 49
        ADDEND=2.0
        GO TO 56
49      IF(INPUT(KK).NE.THREE) GO TO 50
        ADDEND = 3.0
        GO TO 56
50      IF(INPUT(KK).NE. FOUR)  GO TO 51
        ADDEND=4.0
        GO TO 56
51      IF(INPUT(KK).NE.FIVE) GO TO 52
        ADDEND=5.0
        GO TO 56
52      IF(INPUT(KK).NE.SIX) GO TO 53
        ADDEND= 6.0
        GO TO 56
53      IF(INPUT(KK).NE.SEVEN) GO TO 54
        ADDEND= 7.0
        GO TO 56
54      IF(INPUT(KK).NE.EIGHT) GO TO 55
        ADDEND= 8.0
        GO TO 56
55      IF(INPUT(KK).NE.NINE)      GO TO 69
        ADDEND=9.0
56      TEMP=ADDEND+10.0*TEMP
        KK=KK+1
        GO TO 46
69      IF(INPUT(KK).EQ.PERIOD) GO TO 57
        GO TO 68
57      SCALE = 1.0
58      SCALE=0.1*SCALE
        KK=KK+1
        IF(INPUT(KK).NE.ZERO) GO TO 59
        GO TO 58
59      IF(INPUT(KK).NE.ONE) GO TO 60
        TEMP=SCALE+TEMP
        GO TO 58
60      IF(INPUT(KK) .NE. TWO) GO TO 61
        TEMP= 2.0*SCALE+TEMP
        GO TO 58
61      IF(INPUT(KK).NE.THREE)      GO TO 62
        TEMP=3.0*SCALE      +TEMP
        GO TO 58
62      IF(INPUT(KK).NE. FOUR)      GO TO 63
        TEMP=4.0      *SCALE + TEMP

```



```

        GO TO 58
63      IF (INPUT(KK).NE.FIVE) GO TO 64
        TEMP=5.0 *SCALE + TEMP
        GO TO 58
64      IF (INPUT(KK).NE.SIX) GO TO 65
        TEMP=6.0 *SCALE + TEMP
        GO TO 58
65      IF (INPUT(KK).NE.SEVEN) GO TO 66
        TEMP=7.0 *SCALE + TEMP
        GO TO 58
66      IF (INPUT(KK).NE.EIGHT) GO TO 67
        TEMP =8.0 *SCALE + TEMP
        GO TO 58
67      IF (INPUT(KK).NE.NINE) GO TO 68
        TEMP=9.0 *SCALE + TEMP
        GO TO 58
68      KK=KK-1
        CALL JUSTOT(TEMP,INPUT(KL),INPUT(KL+1))
        KL=KL+1
        GO TO 15
200     INPUT(KL)=STRGHD
        LL=0
201     KL=KL+1
        INPUT(KL)=0
202     KK=KK+1
        IF (ISHIFT(INPUT(KK)).EQ.IASTRK) GO TO 204
        INPUT(KL)=IPACK(INPUT(KL),SPACE(6*LL,INPUT(KK)))
        IF (LL.GT.5) GO TO 203
        LL=LL+1
        GO TO 202
203     KK=KK-1
        GO TO 15
204     KK=KK+1
        GO TO 4
80      KL=KL+1
        DO 81 KK=KL,3000
81      INPUT(KK)=IBLANK
        RETURN
        END
SIBFTC DIGITX
        LOGICAL FUNCTION DIGIT(JIG)
        INTEGER ZERO,ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT
        DATA ZERO,ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT,NINE/
1        1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9/
        IF (JIG.EQ.ZERO.OR.JIG.EQ.ONE.OR.JIG.EQ.TWO.OR.JIG.EQ.THREE.OR.
1JIG.EQ.FOUR.OR.JIG.EQ.FIVE.OR.JIG.EQ.SIX.OR.JIG.EQ.SEVEN.OR.
2JIG.EQ.EIGHT.OR.JIG.EQ.NINE) GO TO 8
        DIGIT = .FALSE.
        RETURN
8        DIGIT = .TRUE.
        RETURN
        END
$SIBFTC LETTRX
        LOGICAL FUNCTION LETTER(LET)
        INTEGER A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
        DATA A,B,C,D,E,F,G,H,I,J,K/
12H0A,2H0B,2H0C,2H0D,2H0E,2H0F,2H0G,2H0H,2H0I,2H0J,2H0K/
        DATA L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z/
12H0L,2H0M,2H0N,2H0O,2H0P,2H0Q,2H0R,2H0S,2H0T,2H0U,2H0V,2H0W,
22H0X,2H0Y,2H0Z/
        IF (LET.EQ.A.OR.LET.EQ.B.OR.LET.EQ.C.OR.LET.EQ.D.OR.LET.EQ.E.OR.

```

```

1LET.EQ.F.OR.LET.EQ.G.OR.LET.EQ.H.OR.LET.EQ.I.OR.LET.EQ.J.OR.LET
2.EQ.K.OR.LET.EQ.L.OR.LET.EQ.M.OR.LET.EQ.N.OR.LET.EQ.O.OR.LET.EQ.
3P.OR.LET.EQ.Q.OR.LET.EQ.R.OR.LET.EQ.S.OR.LET.EQ.T.OR.LET.EQ.U.OR.
4LET.EQ.V.OR.LET.EQ.W.OR.LET.EQ.X.OR.LET.EQ.Y.OR.LET.EQ.Z)

```

```

5    GO TO 8
    LETTER = .FALSE.
    RETURN

```

```

8    LETTER=.TRUE.
    RETURN

```

```

    END

```

```

$IBMAP SPACEX 7

```

```

    ENTRY    SPACE

```

```

*    SPACE(BITSTORIGHT,WORDTOBESHIFTED)

```

```

SPACE  CLA*    3,4    FETCH SHIFTING NUMBER

```

```

    STA    *+3

```

```

    CAL*    4,4    STORE S,1-35 IN AC,BITS P,1-35

```

```

    ANA    MASK    AND WORD S,1-35 WITH AC P,1-35

```

```

    ARS    **    SHIFT RIGHT THE AC

```

```

    SLW    MM

```

```

    CLA    MM

```

```

    TRA    1,4

```

```

MM    BSS    1

```

```

MASK  OCT    770000000000    2-7'S,10-0'S

```

```

    END

```

```

$IBMAP IPK

```

```

    ENTRY    IPACK

```

```

*    TAKES THE LOGICAL OR OF TWO ALPHANUMERIC WORDS

```

```

IPACK  CAL*    3,4

```

```

    ORA*    4,4

```

```

    SLW    MM

```

```

    CLA    MM

```

```

    TRA    1,4

```

```

MM    BSS    1

```

```

    END

```

```

$IBMAP JUST

```

```

    ENTRY    JUSTOT

```

```

*    JUSTOT(FL,A,B)=THE ABSOLUTE VALUE OF FL

```

```

*    IS TRANSFORMED TO 2 BCD WORDS AND

```

```

*    STORED IN A AND B

```

```

JUSTOT SXA    BACK,1

```

```

    CLA*    3,4

```

```

    SSP

```

```

    XCA

```

```

    PXA    0,0

```

```

    AXT    6,1

```

```

    ALS    3

```

```

    LGL    3

```

```

    TIX    *-2,1,1

```

```

    STO*    4,4

```

```

    PXA    0,0

```

```

    AXT    6,1

```

```

    ALS    3

```

```

    ALS    3

```

```

    TIX    *-2,1,1

```

```

    STO*    5,4

```

```

BACK  AXT    **,1

```

```

    TRA    1,4

```

```

    END

```

```

$IBMAP ISHFT

```

```

*    SHIFTS THE ARGUMENT RIGHT ONE CHARACTER AND RETURNS VALUE

```

```

    ENTRY    ISHIFT

```

[illegible]

#### Appendix 4. The EULER Polish Interpreter Listing

SCHNEIDER,VI  
EULER

001/68/799

- EFN SOURCE STATEMENT - IFN(S) -

DATE 01/02

```

      INTEGER T, FORMAL, VARIBL, BLKNUM, STORAJ, ASSINE,
      1 BEGIN, TRMNAT, SEMCLN, SYMBOL, COMMA, RSBSCP, CONCAT, TAIL, OUT,
      2 TEST, SWITCH, THEN, FRSTOR, SCNDOR, FSTAND, SCNAND, ADD,
      3 SUBTRC, FLTDIV, EXPNNT, PLUS, TOREAL, TOINTG, TOABSL, PRCCHD,
      4 PRCEND, TRUTH, FALSTY, GREATR, GRATEQ, UNDFND, PRCCHD
      DIMENSION LAVS(2000), LSTRTN(200), IDNTLS(1500), CPRID(500),
      1 IOPRND(500), STORAJ(200), JUMPBK(200), PRAMLS(200)
      COMMON/AVSLX/AVSL, CODE(500)
      INTEGER CODE
      DATA ICOMMA, LEPREN, IRPREN/1H,,6H      (,1H)/
      DATA LOGCAL, ISNMBR, ISLGCL, ISLABL, ISLIST, ISUNDF, ISRFRN/
      16H0.EGCL,6H0.ISNU,6H0.ISLC,6H0.ISLA,6H0.ISLI,6H0.ISUN,6H0.ISRF/
      DATA TRMNAT, BEGIN, SEMCLN, NEW, LABEL, LABDEF, ASSINE, JUMP/
      16H0.END.,6H0.BGIN,3H0.,,6H0.NEW.,6H0.LABL,6H0.LBDF,2H0=,6H0.GOTC/
      DATA CUT, TEST, SWITCH, THEN, NEGATE, FRSTOR, SCNDOR/
      16H0.OUT.,5H0.IF.,6H0.SWCH,6H0.THEN,6H0.NOT.,6H0.FSOR,
      26H0.SCOR/
      DATA LESSEQ, ADD, SUBTRC, PLUS, NEG, MODULO, MLTPLY, INTDIV/
      16H0.LEQ.,2H0+,2H0-,6H0.PLUS,6H0.NEG.,6H0.MDLO,2H0*,4H0././
      DATA FLTDIV, EXPNNT, CONCAT, INSIDE, UNDFND, TAIL, LSCALL, LIST/
      12H0/,3H0**,6H0.CNCT,5H0.IN.,6H0.UNDF,6H0.TAIL,6H0.LSCL,6H0.LIST/
      DATA ISPRCD, ISYMBL, SYMBOL, RSBSCP, VARIBL/
      16H0.ISPR,6H0.ISSY,3H0.*,2H0),6H0.VRBL/
      DATA FSTAND, SCNAND, ISSAME, NOTSAM, GREATR, LESSTN, GRATEQ/
      16H0.FAND,6H0.SAND,5H0.EQ.,6H0.NEQ.,5H0.GT.,5H0.LT.,6H0.GEQ./
      DATA PRCCHD, PRCEND, FORMAL, LABEL, LISTHD,      COMMA, LSTEND/
      13H0.$,3H0$,6H0.FRML,6H0.LABL,3H0.(,2H0,,3H0)./
      DATA NUMBER, TOREAL, LENGTH, TOINTG, TOABSL/
      16H0.NMBR,6H0.REAL,6H0.LNGT,6H0.INTG,6H0.ABST/
      DATA TRUTH, FALSTY/6H0.TRUE,6H0.FALSE/
      CALL INITAS(LAVS,2000)
      DO 1 IK=1,46
      T=11*IK
      I=T-10
      IF(T.GT.500)T=500
      READ(5,2)ITEMP,(CCODE(INDEX),INDEX=I,T)
      2 FORMAT(A1,5X,11A6,8X)
      IF(ISHIFT(ITEMP).EQ.MLTPLY) GO TO 5
      1 CCNTINUE
      5 WRITE(6,9006)(CODE(I),I=1,T)
      WRITE(6,9005)
      I=0
      IK=0
      IBASH=0
      BLKNUM=0
      JMPRTN=0
      T=0
      10 I=I+1
      CALL CUTOOT(7, CODE(I), IBASH)
      ICODE=ISHIFT(CODE(I))
      IF(ICODE .EQ. LSCALL) GO TO 300
      IF(ICODE .EQ. SYMBOL) GO TO 305
      IF(ICODE .EQ. TEST) GO TO 430
      IF(ICODE .EQ. SWITCH) GO TO 440
      IF(ICODE .EQ. NEGATE) GO TO 445

```

SCHNEIDER,VI  
EULER

001/68/799

- EFN SOURCE STATEMENT - IFN(S) -

DATE 01/03

|                       |           |
|-----------------------|-----------|
| IF(ICCDE .EQ. THEN)   | GO TO 10  |
| IF(ICODE .EQ. LIST)   | GO TO 500 |
| IF(ICCDE .EQ. CONCAT) | GO TO 505 |
| IF(ICCDE .EQ. TAIL)   | GO TO 510 |
| IF(ICCDE .EQ. LENGTH) | GO TO 515 |
| IF(ICODE .EQ. LISTHD) | GO TO 600 |
| IF(ICCDE .EQ. COMMA)  | GO TO 605 |
| IF(ICCDE .EQ. LSTEND) | GO TO 615 |
| IF(ICODE .EQ. RSBSCP) | GO TO 650 |
| IF(ICCDE .EQ. BEGIN)  | GO TO 700 |
| IF(ICCDE .EQ. TRMNAT) | GO TO 705 |
| IF(ICODE .EQ. SEMCLN) | GO TO 710 |
| IF(ICODE .EQ. VARIBL) | GO TO 715 |
| IF(ICCDE .EQ. NUMBER) | GO TO 725 |
| IF(ICCDE .EQ. UNDFND) | GO TO 727 |
| IF(ICODE .EQ. NEW)    | GO TO 730 |
| IF(ICODE .EQ. LABDEF) | GO TO 731 |
| IF(ICCDE .EQ. LABEL)  | GO TO 735 |
| IF(ICCDE .EQ. ASSINE) | GO TO 740 |
| IF(ICCDE .EQ. OUT)    | GO TO 755 |
| IF(ICCDE .EQ. FRSTOR) | GO TO 775 |
| IF(ICCDE .EQ. SCNDOR) | GO TO 780 |
| IF(ICODE .EQ. SCNAND) | GO TO 780 |
| IF(ICCDE .EQ. FSTAND) | GO TO 785 |
| IF(ICCDE .EQ. PLUS)   | GO TO 10  |
| IF(ICODE .EQ. NEG)    | GO TO 781 |
| IF(ICODE .EQ. ISSAME) | GO TO 790 |
| IF(ICODE .EQ. NOTSAM) | GO TO 795 |
| IF(ICCDE .EQ. GREATR) | GO TO 796 |
| IF(ICODE .EQ. LESSTN) | GO TO 797 |
| IF(ICCDE .EQ. GRATEQ) | GO TO 798 |
| IF(ICCDE .EQ. LESSEQ) | GO TO 799 |
| IF(ICODE .EQ. ADD)    | GO TO 805 |
| IF(ICODE .EQ. SUBTRC) | GO TO 807 |
| IF(ICCDE .EQ. MLTPLY) | GO TO 810 |
| IF(ICODE .EQ. FLTDIV) | GO TO 811 |
| IF(ICODE .EQ. INTDIV) | GO TO 812 |
| IF(ICODE .EQ. MODULO) | GO TO 813 |
| IF(ICODE .EQ. EXPNNT) | GO TO 816 |
| IF(ICCDE .EQ. TRUTH)  | GO TO 817 |
| IF(ICCDE .EQ. FALSTY) | GO TO 819 |
| IF(ICODE .EQ. LOGCAL) | GO TO 820 |
| IF(ICCDE .EQ. PROCHD) | GO TO 840 |
| IF(ICODE .EQ. INSIDE) | GO TO 845 |
| IF(ICCDE .EQ. PRCEND) | GO TO 865 |
| IF(ICODE .EQ. ISNMBR) | GO TO 860 |
| IF(ICCDE .EQ. ISLGCL) | GO TO 861 |
| IF(ICODE .EQ. ISLABL) | GO TO 862 |
| IF(ICODE .EQ. ISLIST) | GO TO 863 |
| IF(ICODE .EQ. ISUNDF) | GO TO 864 |
| IF(ICODE .EQ. ISPRCD) | GO TO 866 |
| IF(ICODE .EQ. ISYMBL) | GO TO 867 |
| IF(ICCDE .EQ. ISRFRN) | GO TO 868 |
| IF(ICODE .EQ. FORMAL) | GO TO 870 |
| IF(ICCDE .EQ. JUMP)   | GO TO 930 |
| IF(ICODE .EQ. TOREAL) | GO TO 937 |

6

5

4

3

2

SCHNEIDER,VI 001/68/799  
EULER - EFN SOURCE STATEMENT - IFN(S) -

DATE 01/03

IF(ICCODE.EQ.TOINTG) GO TO 938  
IF(ICCODE.EQ.TUABSL) GO TO 939  
WRITE(6,9931)  
STCP

C  
C ASSUMES VARIBL HAS PLACED A LINK TO THE DATUM IN CPRND(IK)  
C

300 LINK = LNKR(OPRND(IK))  
IF(ID(INHALT(LINK)).NE.6) GO TO 886C  
IOPRNC(IK) = 6  
CALL STRDIR(CENT(LINK+1),OPRND(IK))  
GO TO 10

305 I=I+1  
IK=IK+1  
IOPRNC(IK)=7  
CALL STRDIR(CODE(I),OPRND(IK))  
GO TO 10

C  
C TEST CPERAND FOR TRUTH. IF FALSE, SCAN AHEAD UNTIL A MATCHING  
C THEN IS FOUND. IF TRUE, CONTINUE EVALUATION.  
C

430 IF(CPRND(IK).EQ.1.0) GO TO 10  
ITT=0

431 I=I+1  
ICCODE=ISHIFT(CODE(I))  
IF(ICCODE.EQ.TEST) ITT=ITT+1  
IF(ICCODE.EQ.THEN) GO TO 432  
GO TO 431

432 IF(ITT.EQ.0) GO TO 10  
ITT=ITT-1  
GO TO 431

C  
C EXCHANGE TOPMOST OPERANDS DURING A CONDITIONAL STATEMENT.  
C

440 TEMP=OPRND(IK-1)  
ITT=IOPRND(IK-1)  
OPRND(IK-1)=CPRND(IK)  
IOPRND(IK-1)=IOPRND(IK)  
CPRND(IK)=TEMP  
IOPRND(IK)=ITT  
GO TO 10

C  
C NEGATE TOPMOST OPERAND FOR CONDITIONAL STATEMENT AND LOGICAL  
C NEGATION.  
C

445 IF(IOPRND(IK).NE.1) GO TO 8775  
OPRND(IK)=1.0-OPRND(IK)  
GO TO 10

C  
C CREATION OF A LIST OF LENGTH OPRND(IK).  
C

500 IF(IOPRND(IK).NE.0) GO TO 8806  
INDEX=OPRND(IK)  
ICPRNC(IK)=3  
IF(INDEX.NE.0) GO TO 501  
CALL STRDIR(0,CPRND(IK))

SCHNEIDER,VI 001/68/799  
EULER - EFN SOURCE STATEMENT - IFN(S) -

DATE 01/03

501 GO TO 10  
IVRBL=NUCELL(DUMMY)  
IF(INDEX.EQ.1) GO TO 503  
LLCELL=IVRBL  
DC 502 ITEMP=2, INDEX  
LRCELL=NUCELL(DUMMY)  
CALL SETIND(4,-1,LRCELL,LLCELL)  
C THE NUCELL ROUTINE ZEROES THE LNKR FIELDS.  
502 LLCELL=LRCELL  
503 CALL SETDIR(0,0,IVRBL,OPRND(IK))  
GO TO 10

C  
C CONCATENATION OF TWO LISTS. A LINK TO THE RESULTING LIST IS  
C RETURNED TO THE TOP OF THE OPRND STACK. THE TWO CONCATENATED LISTS  
C ARE COPIED BY THE COPY ROUTINE. THE COPY ROUTINE HAS PROVISIONS  
C FOR LISTS THAT HAVE ZERO REFERENCE COUNT (I.E., NO NAME) AND FOR  
C EMPTY LISTS.  
C

505 IF(IOPRND(IK).NE.3.OR.IOPRND(IK-1).NE.3) GO TO 8510  
LINKO=LNKR(OPRND(IK-1))  
CALL COPY(LINKO,ITOP,IBOT)  
LINKO=LNKR(OPRND(IK))  
CALL COPY(LINKO,JTOP,JBOT)  
IK=IK-1  
CALL STRDIR(ITOP,OPRND(IK))  
CALL SETIND(-1,-1,JTOP,IBOT)  
GO TO 10

C  
C TAKES THE SUFFIX OF A LIST, IGNORES REFERENCE COUNT OF  
C FIRST CELL, AND PUTS LINK TO THAT CELL ON TOP OF OPERAND LIST  
C

510 ITEMP=LNKR(OPRND(IK))  
IF(IOPRND(IK).NE.3) GO TO 8510  
IF(ITEMP.EQ.0) GO TO 8511  
ITEMP=LNKR(INHALT(ITEMP))  
IF(ITEMP.EQ.0) GO TO 8650  
CALL SETDIR(ITEMP,OPRND(IK))  
CALL SETIND(-1,LNKL(INHALT(ITEMP))+1,-1,ITEMP)  
GO TO 10

C  
C FINDS THE LENGTH OF THE LIST WHOSE POINTER IS AN OPERAND.  
C

515 IF(IOPRND(IK).NE.3) GO TO 8510  
TEMP=0.0  
ITEMP=LNKR(OPRND(IK))  
ICPRND(IK)=0  
516 IF(ITEMP.LE.0) GO TO 517  
TEMP=TEMP+1.0  
ITEMP=LNKR(INHALT(ITEMP))  
GO TO 516  
517 OPRND(IK)=TEMP  
GO TO 10

C  
C A PLACE IS RESERVED ON THE OPERAND LIST FOR A LINK.  
C

600 IK=IK + 1



SCHNEIDER, VI 001/68/799  
EULER - EFN SOURCE STATEMENT - IFN(S) -

DATE 01/79

IOPRND(IK) = 3  
CALL STRDIR(LISTHD, OPRND(IK))  
GO TO 10

C  
C TYPED INFORMATION ON TOP OF THE OPERAND LIST IS REPLACED BY  
C AN INTEGER LINK TO A LIST CELL THAT CONTAINS  
C THIS INFORMATION. THE IK COUNT IS NOT DECREMENTED AND THE  
C SUBLIST REFERENCE COUNTERS ARE INCREMENTED.

C  
605 IKK=IOPRND(IK)  
IJJ=LNKR(OPRND(IK))  
IF(IKK.NE.3.OR.IJJ.LE.0) GO TO 610  
CALL SETIND(-1, LNKL(INHALT(IJJ))+1, -1, IJJ)  
610 ITEMP=NUCELL(DUMMY)  
CALL SETIND(IKK, 0, 0, ITEMP)  
CALL STRIND(OPRND(IK), ITEMP+1)  
CALL STRDIR(ITEMP, OPRND(IK))  
IOPRND(IK)=3  
GO TO 10

C  
C PERFORMS THE COMMA OPERATION ON TOPMOST DATUM IN OPERAND STACK.  
C THEN PROCEEDS DOWN THE OPERAND STACK TO LINK TOGETHER OPERANDS  
C UNTIL LISTHEAD \*\*. (\*\* IS ENCOUNTERED AS AN OPERAND. FINALLY, CHECKS  
C TO SEE IF THE RESULTING LIST IS THE PARAMETERLIST OF A PROCEDURE  
C CALL.

C  
615 IF(INTRSC(OPRND(IK), LISTHD).NE.0) GO TO 620  
CALL STRDIR(0, OPRND(IK))  
GO TO 10

620 ITEMP=NUCELL(DUMMY)  
IKK=IOPRND(IK)  
IJJ=LNKR(OPRND(IK))  
IF(IKK.NE.3.OR.IJJ.LE.0) GO TO 630  
CALL SETIND(-1, LNKL(INHALT(IJJ))+1, -1, IJJ)  
630 CALL SETIND(IKK, 0, 0, ITEMP)  
CALL STRIND(OPRND(IK), ITEMP+1)  
CALL STRDIR(ITEMP, OPRND(IK))  
IOPRND(IK)=3

645 IF(INTRSC(OPRND(IK-1), LISTHD).EQ.0) GO TO 646  
CALL SETIND(-1, 0, LNKR(OPRND(IK)), LNKR(OPRND(IK-1)))  
IK=IK-1  
GO TO 645

646 IK=IK-1  
OPRND(IK)=OPRND(IK+1)  
IF(IOPRND(IK-1).NE.6) GO TO 10

C  
C THE FORMAL ROUTINE EXPECTS TO FIND A LINK TO THE FIRST  
C CELL OF AN ID-DATUM PAIR ON PRAMLS. FORMAL CHECKS FOR  
C LISTEND AND PARAMETER MATCHING. THE PARAMETER LIST IS LOCAL  
C TO THE CALLING BLOCK OF THE PROCEDURE AND IS DEMOLISHED BY END.

JMPRTN=JMPRTN+1

JUMPEK(JMPRTN)=1

T=T+1

ITT=3\*T

CALL STRDIR(OPRND(IK), PRAMLS(JMPRTN))

CALL STRDIR(OPRND(IK), IDNTLS(ITT))

SCHNEIDER,VI 001/68/799  
EULER - EFN SOURCE STATEMENT - IFN(S) -

DATE 01/03

CALL SETDIR(3,1,0,IDNTLS(ITT-1))  
I=OPRND(IK-1)  
IK=IK-2  
BLKNUM=BLKNUM+1  
STORAJ(BLKNUM)=T+1  
GO TO 10

C  
C THE RIGHT BRACKET ACTS AS AN OPERATOR SO THAT THE TOPMOST OPERAND  
C SPECIFIES THE CELL OF A LIST WHOSE LINK IS THE NEXT OPERAND  
C DOWN. A REFERENCE (LINK) IS RETURNED TO THE NEXT OPERAND DOWN,  
C PRESERVES THE BLOCKNUMBER FOUND BY VARIBL.  
C

650 INDEX=OPRND(IK)  
IK=IK-1  
IF(INDEX.LE.0) GO TO 10  
LINK=LNKR(INHALT(1+LNKR(OPRND(IK))))  
651 IF(LINK.LT.2) GO TO 8650  
INDEX=INDEX-1  
IF(INDEX.EQ.0) GO TO 652  
LINK=LNKR(INHALT(LINK))  
GO TO 651  
652 CALL SETDIR(-1,-1,LINK,OPRND(IK))  
GO TO 10

C

700 BLKNUM = BLKNUM + 1  
STORAJ(BLKNUM) = T+1  
GO TO 10

C THIS IS THE BLOCK END ROUTINE. ITS MAIN FEATURE IS A SEARCH  
C FOR LISTS TO RETURN TO LAVS. THE BLOCKNUMBER IS DECREMENTED,  
C AND, IF EQUAL TO ZERO, THE PROGRAM TERMINATES.

C

705 INDEX = STORAJ(BLKNUM)  
IF(T.LT.INDEX) GO TO 709  
706 LLNK=LNKR(IDNTLS(3\*T))  
IF(ID(IDNTLS(3\*T-1)).NE.3.OR.LLNK.LE.0)GO TO 708  
CALL ERASE(LLNK)  
708 ITT=3\*T  
IDNTLS(ITT)=0  
IDNTLS(ITT-1)=0  
IDNTLS(ITT-2)=0  
T = T - 1  
IF(T.GE.INDEX) GO TO 706  
709 BLKNUM = BLKNUM - 1  
IF(BLKNUM .GT. 0) GO TO 10  
STOP

C

C SEMICOLON REMOVES THE TOPMOST OPERAND OF THE OPERAND LIST.  
C IF THIS OPERAND IS A LINK TO SOME LIST WHOSE FIRST CELL HAS  
C ZERO REFERENCE COUNT ERASE IS CALLED.  
C

710 IF(IOPRND(IK) .NE. 3) GO TO 713  
LLNK = LNKR(OPRND(IK))  
IF(LLNK.LE.0) GO TO 713  
LCOUNT=LNKL(INHALT(LLNK))  
IF(LCOUNT .GT. 0) GO TO 713  
CALL ERASE (LLNK)

SCHNEIDER, VI  
EULFK

001/68/799

- EFN SOURCE STATEMENT - IFN(S) -

DATE 01/03

713 IK = IK - 1  
GO TO 10

C  
C IT IS ASSUMED THAT LABELS AND REFERENCES ARE STORED IN THE LNK  
C FIELDS OF WORDS AND THAT THEIR DYNAMIC BLOCKNUMBER IS STORED IN  
C THE LKFL FIELDS. THE NAME FOLLOWING THE VARIBL COMMAND IS LOOKED  
C UP ON THE IDENT STACK, AND A LINK TO IDENT(3\*T'-1), FOR SOME T', IS  
C PLACED ON TOP OF THE OPRND STACK.

715 I = I + 1  
ITEMP = T  
IVRBL = CODE(I)

716 IF (INTRSC(IVRBL, IDNTLS(3\*ITEMP-2)).EQ.0) GO TO 717  
ITEMP = ITEM - 1  
IF (ITEMP .LE. 0) GO TO 8716  
GO TO 716

717 IK = IK + 1  
LLNK=MADOV(IDNTLS(3\*ITEMP-1))  
CALL SETDIR(6, LNK(INHALT(LLNK)), LLNK, OPRND(IK))  
IOPRND(IK) = 5  
GO TO 10

C  
725 I=I+2  
IK=IK+1  
CALL JUSTIN(OPRND(IK), CODE(I-1), CODE(I))  
IGPRND(IK) = 0  
GO TO 10  
727 IK=IK + 1  
OPRND(IK) = 0.0  
IGPRND(IK) = 4  
GO TO 10

C  
730 I = I + 1  
T = I + 1  
ITT=3\*T  
IDNTLS(ITT-2)=CODE(I)  
CALL SETDIR(4,0,BLKNUM, IDNTLS(ITT-1))  
IDNTLS(ITT)=0  
GO TO 10

731 I=I + 1  
GO TO 10

C  
C THIS ROUTINE SEARCHES FOR THE LOCATION OF THE  
C LABEL, THEN PLACES THE LINK TO THIS LABEL, EXPRESSED AS A  
C VALUE OF I, INTO IDNTLS(T, 3) ALONG WITH ITS BLOCKNUMBER.

735 T=T + 1  
ITT=3\*T  
I=I + 1  
KLBL = CODE(T)  
IDNTLS(ITT-2)=KLBL  
CALL SETDIR(2,0,BLKNUM, IDNTLS(ITT-1))  
INDEX = I  
ITEMP = 0

736 INDEX = INDEX + 1  
ICODE=ISHIFT(CODE(INDEX))

SCHNEIDER,VI 001/68/799  
EULER - EFN SOURCE STATEMENT - IFN(S) -

DATE 01/02

```

      IF(ICCDE.EQ.BEGIN) ITEMP=ITEMP+1
      ICODE=ISHIFT(CODE(INDEX))
      IF(ICCDE.EQ.TRMNAT) ITEMP=ITEMP-1
      IF(ITEMP.LT. 0) GO TO 8736
      IF(ICODE.NE.LABDEF)GO TO 736
      INDEX = INDEX + 1
      IF(CCODE(INDEX).NE.KLBL .OR. ITEMP.NE.0) GO TO 736
      INDEX = INDEX + 1
737  IF(ISHIFT(CODE(INDEX)).NE.LABDEF)GO TO 738
      INDEX = INDEX + 2
      GO TO 737
738  CALL SETDIR(0,BLKNUM,INDEX,IDNTLS(ITT))
      GO TO 10

```

C  
C THE ASSIGNEE IS A REFERENCE (TYPE 5) BECAUSE OF THE UNIFORM  
C CONVENTIONS OF TREATING THE SEQUENCE \*\*.VRBL.\*NAME\*\*\* AND THE  
C RESULT OF USING RSBSCP ON A LIST FOR SUBSCRIBING BOTH AS  
C REFERENCE POINTERS. THE LINKLEFT FIELD OF THE ASSIGNEE OPERAND  
C IS ITS DYNAMIC BLOCKNUMBER, AND THE LINKRIGHT FIELD POINTS TO THE  
C IDNTLS OR LIST CELL THAT PRECEDES ITS DATUM IN COMPUTER MEMORY.  
C IF THE ASSIGNED OPERAND IS A REFERENCE OR A LABEL, ITS LINKLEFT  
C FIELD ALSO CONTAINS THE DYNAMIC BLOCKNUMBER. IN THIS CASE, THE  
C ASSIGNMENT STATEMENT WILL NOT BE EXECUTED IF THE ASSIGNEE IS IN A  
C BLOCK GLOBAL TO THE BLOCK OF THE ASSIGNED.

C  
740 IF(IOPRND(IK-1) .NE. 5) GO TO 8740  
INDEX = IOPRND(IK)  
LRNK = LNK(OPRND(IK))  
LINK = LNK(OPRND(IK- 1))  
IF(INDEX .EQ. 5 .OR. INDEX .EQ. 2) GO TO 745  
IF(INDEX.NE.3.OR.LRNK.LE.0) GO TO 741  
CALL SETIND(-1, LNK(INHALT(LRNK)) + 1, -1, LRNK)  
C LOCAL AND GLOBAL LISTS CAN BE ASSIGNED TO ONE ANOTHER  
CALL SETDIR(0,0,-1,OPRND(IK))  
741 IF(ID(INHALT(LINK)).NE.3.OR.LNK(INHALT(LINK+1)).LE.0)GO TO 742  
CALL ERASE(INHALT(LINK + 1))  
742 CALL SETIND(INDEX, -1, -1, LINK)  
CALL STRIND(OPRND(IK), LINK + 1)  
OPRND(IK - 1) = OPRND(IK)  
IOPRND(IK - 1) = IOPRND(IK)  
IK = IK - 1  
GO TO 10  
745 IF(LNKL(OPRND(IK)) .GT. LNKL(OPRND(IK - 1))) GO TO 8745  
GO TO 741

C  
C THIS IS THE WRITEOUT ROUTINE. THE TOP DATUM ON THE OPRND STACK IS  
C WRITTEN OUT, AND THIS DATUM IS LEFT UNTOUCHED ON THE OPRND STACK.  
C

```

755  WRITE(6,774)
      ITT=0
      ITEMP=IOPRND(IK)
      IF(ITEMP.NE.3) GO TO 761
C 761 IS THE LABEL CORRESPONDING TO A NON-LIST OUTPUT.
      IF(LNK(OPRND(IK)).GT.0) GO TO 756
C AN EMPTY LIST HAVING ZERO LINK IN O(IK) IS TREATED SEPARATELY.
      WRITE(6,770)

```

SCHNEIDER,VI  
EULER

001/68/799

- EFN SOURCE STATEMENT - IFN(S) -

DATE 01/03

```

WRITE(6,774)
IBASH=0
GO TO 10
756 LSTCNT=0
LLNK=LNKR(OPRND(IK))
C LEFT PARENTHESIS OF OUTER LIST IS WRITTEN.
CALL OUTOUT(7,LFPREN,ITT)
757 INLNK=INHALT(LLNK)
IF(ID(INLNK).NE.3)GO TO 758
C LSTCNT GIVES DEPTH INTO LIST STRUCTURE WRITTEN OUT.
LSTCNT=LSTCNT+1
C DESCENT INTO A SUBLIST
LSTRTN(LSTCNT)=LLNK
LLNK=LNKR(INHALT(LLNK+1))
CALL OUTOUT(7,LFPREN,ITT)
GO TO 757
C PRINTOUT OF A DATUM. CHARACTERS NOT SEPARATED BY COMMAS HERE.
758 CALL OUTOUT(ID(INLNK),INHALT(LLNK+1),ITT)
IF(ID(INLNK).EQ.7) GO TO 759
IF(LNKR(INLNK).GT.0) CALL OUTOUT(7,ICOMMA,ITT)
759 IF(LNKR(INLNK).GT.0) GO TO 762
760 CALL OUTOUT(7,IRPREN,ITT)
IF(LSTCNT.LE.0) GO TO 763
C RISE FROM SUBLIST TO MASTER LIST.
LLNK=LNKR(INHALT(LSTRTN(LSTCNT)))
LSTCNT=LSTCNT-1
IF(LLNK.EQ.0) GO TO 760
CALL OUTOUT(7,ICOMMA,ITT)
GO TO 757
C WRITEOUT OF NEXT ELEMENT OF A LIST.
762 LLNK=LNKR(INLNK)
GO TO 757
C END OF LIST WRITEOUT.
763 WRITE(6,774)
IBASH=0
GO TO 10
C WRITEOUT OF NORMAL NONLIST ELEMENT.
761 ICCNTR=2
CALL OUTOUT(ITEMP,OPRND(IK),ICONTR)
WRITE(6,774)
IBASH=0
GO TO 10
770 FORMAT(1H0,1X,3H( ))
774 FORMAT(1H0/)
C
C THIS ROUTINE CHECKS THE TOPMOST OPERAND FOR TRUTH OR FALSETY.
C IF TRUE, SKIPS TO SCNDOR+1. IF FALSE, DELETES OPERAND
C AND PROCEEDS TO EVALUATE SECOND OPERAND. SECONDOR
C FINDS AN OPERAND THAT IT LEAVES UNTOUCHED.
C
775 IF(IOPRND(IK).NE.1) GO TO 8775
776 IF(OPRND(IK).EQ.1.0) GO TO 777
IK=IK-1
GO TO 10
777 ITT=0
778 I=I+1

```

SCHNEIDER,VI  
EULER

001/68/799

- EFN SOURCE STATEMENT - IFN(S) -

DATE 01/03

```

      ICODE=ISHIFT(CODE(I))
      IF(ICODE.EQ.          FRSTOR)ITT=ITT+1
      IF(ICODE.EQ.          SCNDOR)GO TO 779
      GO TO 778

```

```

779      IF(ITT.EQ.0) GO TO 10
      ITT=ITT-1
      GO TO 778

```

C

```

780      IF(IOPRND(IK).NE.1) GO TO 8775
      GO TO 10

```

C

C THIS ROUTINE CHECKS TOPMOST OPERAND FOR TRUTH OR FALSETY.

C IF FALSE, SKIPS TO SCNAND+1. IF TRUE, DELETES OPERAND

C AND PROCEEDS TO EVALUATE SECOND OPERAND.

C

```

785      IF(IOPRND(IK).NE.1) GO TO 8775
786      IF(OPRND(IK).EQ.0.0) GO TO 787
      IK=IK-1
      GO TO 10

```

```

787      ITT=0

```

```

788      I=I+1

```

```

      ICODE=ISHIFT(CODE(I))

```

```

      IF(ICODE.EQ.          FSTAND)ITT=ITT+1

```

```

      IF(ICODE.EQ.          SCNAND)GO TO 789

```

```

      GO TO 788

```

```

789      IF(ITT.EQ.0) GO TO 10

```

```

      ITT=ITT-1

```

```

      GO TO 788

```

C

C A LINK IS PLACED TO PROCHD. SKIP TO PROCEND + 1.

C

```

840      IK = IK+1
      OPRND(IK) = I
      IOPRND(IK) = 6
      INDEX = 0
841      I = I+1

```

```

      ICODE=ISHIFT(CODE(I))

```

```

      IF(ICODE.EQ.          PROCHD)INDEX=INDEX+1

```

```

      IF(ICODE.EQ.          PROCEND)GO TO 842

```

```

      GO TO 841

```

```

842      IF(INDEX.LE.0) GO TO 10

```

```

      INDEX = INDEX-1

```

```

      GO TO 841

```

C

C PROVIDES A RETURN JUMP AND DEMOLITION OF BLOCK STORAGE VIA  
C THE ORDINARY END ROUTINE.

C ALSO DESTROYS PARAMETER LIST OF THE PROCEDURE IF IT HAS ONE.

C

```

865      I = JUMPBK(JMPRTN)
      JMPRTN = JMPRTN - 1

```

```

      GO TO 705

```

```

705 IS WHERE THE END ROUTINE IS.

```

C

C INITIATES PROCEDURE CALLS HAVING NO PARAMETERS AND FETCHES DATUM  
C OF NON-PROCEDURE NAMES INTO OPRND(IK).

C

SCHNEIDER,VI  
EULER

001/68/799

- EFN SOURCE STATEMENT - IFN(S) -

DATE 01/03

```

845  LINK = LNKR(OPRND(IK))
      ICID = ID(INHALT(LINK))
      IOPRNC(IK) = IDID
      IF(IDID.NE. 6) GC TO 850
      IK = IK - 1
      JMPRTN = 1 + JMPRTN
      JUMPRK(JMPRTN) = I
      PRAMLS(JMPRTN)=0.0
      BLKNUM = 1 + BLKNUM
      STORA(BLKNUM) = 1 + T
      I = CNT(LINK + 1)
      IF(ISHIFT(CODE(I+1)).EQ.FORMAL)GO TO 8845

```

```

      GO TO 10
850  CALL STRDIR(INHALT(LINK+1),OPRND(IK))
      GO TO 10

```

C  
C FORMAL CALLS FOR THE LINK OF A LIST OF PARAMATERS STORED ON TOP  
C OF PRAMLS.IF THIS LIST OF PARAMETERS IS TOO SHORT THE UNMATCHED  
C PROCEDURE PARAMETERS ARE UNDEFINED. THIS PARAMETER LIST HAS BEEN  
C MADE LOCAL TO THE CALLING BLOCK BY LSTEND.NO REFERENCE COUNTERS  
C NEED TO BE INCREMENTED FOR ANY POSSIBLE LIST PARAMETERS, SINCE THE  
C LIST CONSTRUCTION HAS ALREADY DONE THIS.

```

870  LINK=LNKR(PRAMLS(JMPRTN))
      I=I+1
      T=T+1
      CALL STRDIR(CODE(I),IDNTLS(3*T-2))
      IF(LINK.EQ. C) GO TO 875
      CALL STRDIR(INHALT(LINK),INHLENK)
      CALL STRDIR(INHLENK,IDNTLS(3*T-1))
      IF(ID(INHLENK).NE.2.AND.ID(INHLENK).NE.5)GO TO 872
      CALL SETDIR(-1,-1,LNKL(INHALT(LINK+1)),IDNTLS(3*T-1))
C    THE BLOCK NUMBER OF THE REFERENCE OR LABEL IS BROUGHT TO
C    IDNTLS(T, 2)

```

```

872  CALL STRDIR(INHALT(LINK+1),IDNTLS(3*T))
      CALL STRDIR(INHLENK,PRAMLS(JMPRTN))

```

```

      GO TO 10
875  IDNTLS(3*T)=0
      CALL SETDIR(4,0,0,IDNTLS(3*T-1))
      GC TO 10

```

C  
805 OPRND(IK-1)=OPRND(IK-1)+OPRND(IK)  
806 IF(IOPRND(IK-1).NE.0.OR.IOPRND(IK).NE.0) GO TO 8806  
 IK=IK-1

```

      GO TO 10
807  OPRND(IK-1)=OPRND(IK-1)-OPRND(IK)
      GO TO 806

```

```

810  OPRND(IK-1)=OPRND(IK-1)*OPRND(IK)
      GO TO 806

```

```

811  IF(OPRND(IK).EQ.0.0) GO TO 8811
      OPRND(IK-1)=OPRND(IK-1)/OPRND(IK)
      GC TO 806

```

```

812  IF(OPRND(IK).EQ.0.0) GO TO 8811
      ITEMP=OPRND(IK-1)/OPRND(IK)
      OPRND(IK-1)=ITEMP
      GC TO 806

```

SCHNEIDER, VI  
EULER

001/68/799

- EFN SOURCE STATEMENT - IFN(S) -

DATE 01/03

```

813  R2=OPRND(IK)
      IF(R2.EQ.0.0) GO TO 8811
      OPRND(IK-1)=AMOD(OPRND(IK-1),R2)
      GC TO 806
C
816  OPRND(IK-1)=OPRND(IK-1)**OPRND(IK)
      GO TO 806
C
790  IF(IOPRND(IK).EQ.7) GO TO 310
      IF(OPRND(IK-1)-OPRND(IK))791,793,791
791  OPRND(IK-1) = 0.0
792  IF(IOPRND(IK-1).NE.0.OR.IOPRND(IK).NE.0) GO TO 8792
      IK = IK-1
      IOPRND(IK) = 1
      GO TO 10
310  IF(IOPRND(IK-1).NE.7)GO TO 8310
      IF(INTRSC(OPRND(IK-1),OPRND(IK)))791,793,791
793  OPRND(IK-1) = 1.0
      GO TO 792
C
795  IF(OPRND(IK-1)-OPRND(IK)) 793,791,793
796  IF(OPRND(IK-1)-OPRND(IK)) 791,791,793
797  IF(OPRND(IK-1)-OPRND(IK)) 793,791,791
798  IF(OPRND(IK-1)-OPRND(IK)) 791,793,793
799  IF(OPRND(IK-1)-OPRND(IK)) 793,793,791
C
860  IF(ID(INHALT(LNKR(OPRND(IK)))).EQ.0) GO TO 871
      GO TO 873
861  IF(ID(INHALT(LNKR(OPRND(IK)))).EQ.1) GO TO 871
      GO TO 873
862  IF(ID(INHALT(LNKR(OPRND(IK)))).EQ.2) GO TO 871
      GO TO 873
863  IF(ID(INHALT(LNKR(OPRND(IK)))).EQ.3) GO TO 871
      GO TO 873
864  IF(ID(INHALT(LNKR(OPRND(IK)))).EQ.4) GO TO 871
      GO TO 873
866  IF(ID(INHALT(LNKR(OPRND(IK)))).EQ.6) GO TO 871
      GO TO 873
867  IF(ID(INHALT(LNKR(OPRND(IK)))).EQ.7) GO TO 871
      GO TO 873
868  IF(ID(INHALT(LNKR(OPRND(IK)))).EQ.5) GO TO 871
      GO TO 873
C
871  OPRND(IK)=1.0
      IOPRND(IK)=1
      GO TO 10
873  OPRND(IK)=0.0
      IOPRND(IK)=1
      GO TO 10
C
817  OPRND(IK)=1.0
818  IOPRND(IK)=1
      IK=IK+1
      GO TO 10
819  OPRND(IK)=0.0
      GO TO 818

```

6

5

4

3

2



SCHNEIDER, VI 001/68/799  
 EULER - EFN SOURCE STATEMENT - IFN(S) -

DATE 01/

C

820 IF(CPRND(IK).GT.0.0) GO TO 871  
 GO TO 873

C

930 IF(IOPRND(IK).NE.2) GO TO 8930  
 I=LNKR(OPRND(IK))-1  
 JMPRTN= JMPRTN+LNKL(OPRND(IK))-BLKNUM  
 IF(JMPRTN.LT.0) JMPRTN=0  
 IK=0  
 GO TO 10

C

937 IOPRND(IK)=0  
 GO TO 10

938 ITEMP=OPRND(IK)  
 OPRND(IK)=ITEMP  
 GO TO 10

939 IF(OPRND(IK).GE.+0.0) GO TO 10

781 IF(IOPRND(IK).NE.0) GO TO 8806  
 OPRND(IK)=-OPRND(IK)  
 GO TO 10

8310 WRITE(6,9310)  
 STOP

8511 WRITE(6,9511)  
 STOP

8832 WRITE(6,9832)ITEMP  
 STOP

8930 WRITE(6,9930) I  
 STOP

8716 WRITE(6,9716) IVRBL  
 STOP

8845 WRITE(6,9845)CODE(I),CODE(I+1)  
 STOP

8740 WRITE(6,9740)I  
 STOP

8745 ITEMP = INHALT(LNKR(OPRND(IK - 1)) - 1)  
 WRITE(6,9745) ITEMP  
 STOP

8650 WRITE(6,9650)  
 STOP

8775 WRITE(6,9775)  
 STOP

8806 WRITE(6,9806)  
 STOP

8811 WRITE(6,9811)  
 STOP

8736 WRITE(6,9736)KLBL  
 STOP

8792 WRITE(6,9792)  
 STOP

8860 WRITE(6,9860)  
 STOP

8510 WRITE(6,9510)  
 STOP

9005 FORMAT(1H0,//////////)

9006 FORMAT(1H0,5X,12A6)

9310 FORMAT(1H0,52HATTEMPTED COMPARISON OF CHARACTER AND NONCHARACT

SCHNEIDER, VI  
EULER

001/68/799

- EFN SOURCE STATEMENT - IFN(S) -

DATE 01/01

1ER. )

9930 FORMAT(37HNO LABEL SUPPLIED TO JUMP INSTRUCTION,5X,A6)  
9931 FORMAT(37HILLEGAL COMMAND OR OPERAND. TERMINATE)  
9510 FORMAT(33HLIST OPERATION ON NONLIST OPERAND)  
9511 FORMAT(20HSUFFIX OF EMPTY LIST)  
9650 FORMAT(46HSUBSCRIPT CALLED FOR LIST CELLS NOT YET EXTANT)  
9716 FORMAT(12HREFERENCE TO,2X,A6,2X,24HAN UNDECLARED IDENTIFIER)  
9736 FORMAT( A6,1X,35HIS NOT USED AS A LABEL IN ITS BLOCK)  
9740 FORMAT(8HAT CODE(,16,35H), ASSIGNMENT OF VALUE TO VALUE. )  
9745 FORMAT(45HASSIGNMENT OF LOCAL REFERENCE TO GLOBAL IDENT,2X,A6)  
9775 FORMAT(47HATTEMPTED LOGICAL OPERATION, NONLOGICAL OPERAND)  
9792 FORMAT(45HRELATION SCUGHT BETWEEN NONNUMERICAL OPERANDS)  
9806 FORMAT(43HARITHMETIC ATTEMPT ON NONNUMERICAL OPERANDS)  
9811 FORMAT (26HDIVISION BY ZERO ATTEMPTED)  
9832 FORMAT(A6,36HLABEL NOT DECLARED AT HEAD OF BLOCK.)  
9845 FORMAT(38HPARAMATERLESS PROCEDURE CALL EXPECTED.,2X,A6,A6)  
9860 FORMAT(48HPROCEDURE CALL ON NONPROCEDURE VARIABLE AND LIST)

END

SCHNEIDER,VI 001/68/799  
INITX - EFN SOURCE STATEMENT - IFN(S) -

DATE 01,

SUBROUTINE INITAS(M,N)

C

C

C

THIS SUBROUTINE INITIALIZES THE STRUCTURE OF THE DIMENSIONED  
ARRAY M INTO A LIST OF AVAILABLE SPACE WITH READER CELL AVSL.

DIMENSION M(N)

COMMON/AVSLX/AVSL, CODE(500)

DO 2 I = 1,N

2

M(I) = 0

K = N-2

DO 3 I=1,K,2

3

CALL SETDIR(-1,-1,MADOV(M(I+2)),M(I))

CALL SETDIR(0,MADOV(M(N-1)),MADOV(M(1)),AVSL)

RETURN

END

C

SCHNEIDER,VI 001/68/799  
ERASEX - EFN SOURCE STATEMENT - IFN(S) -

DATE 01/03

## SUBROUTINE ERASE(LINK)

C  
C RETURNS FIRST CELL OF LIST TO LAVS IF REFERENCE COUNT IS ZERO  
C OR ONE. OTHERWISE, DECREASES REFERENCE COUNT OF THE FIRST CELL.  
C

COMMON/AVSLX/AVSL, CODE(500)  
ILINK = INHALT(LINK)  
IF(LNK(LINK).GT.1) GO TO 2060  
CALL SETIND(-1, -1, LINK, LNKL(AVSL))  
CALL SETDIR(-1, IBTM(LINK, +1), -1, AVSL)

RETURN

2060 CALL SETIND(-1, LNKL(ILINK)-1, -1, LINK)

RETURN

END

SCHNEIDER, VI  
IBTMX

001/68/799

- EFN SOURCE STATEMENT - IFN(S) -

DATE 01/

## FUNCTION IBTM(LLNK, IFLAG)

```
C
C  IF IFLAG=-1, FINDS THE LAST CELL OF A NONEMPTY LIST WHOSE FIRST
C  CELL HAS ADDRESS GIVEN BY LLNK. IF IFLAG=+1, RETURNS THE LAST LIST
C  CELL NOT REFERENCED BY ANOTHER LIST NAME.
C
      LLNR=LLNK
2000  LLNN=LLNR
      LLNR=LNR(INHALT(LLNN))
      IF(IFLAG)2002,3000,2001
2001  IF(LNKL(INHALT(LLNR)))3000,2002,2005
2002  IF(LLNR)3000,2005,2000
2005  IBTM=LLNN
      RETURN
3000  WRITE(6,3005)
3005  FORMAT(1H0,14HERROR IN IBTM. )
      STOP
      ENC
```

SCHNEIDER,VI 001/68/799  
NUCELX - EFN SOURCE STATEMENT - IFN(S) -

DATE 01/03

## FUNCTION NUCELL(X)

```
C
C GETS A NEW CELL FROM LAVS, IF THAT CELL HAS A SUBLIST,
C RETURNS THE SUBLIST TO LAVS BY CALLING ERASE.
C
      COMMON/AVSLX/AVSL, CODE(500)
      M=LNKR(AVSL)
      IF(M.GT.0) GO TO 1
2     WRITE(6,901)
      STOP
1     MM=INHALT(M)
      IF(ID(MM).NE.3.OR.LNKR(INHALT(M+1)).EQ.0) GO TO 3
4     CALL ERASE(LNKR(INHALT(M+1)))
3     CALL SETDIR(-1,-1,LNKR(MM),AVSL)
      CALL STRIND(0,M)
      CALL STRIND(0,M+1)
      NUCELL = M
      RETURN
901 FORMAT(1H1,6X,33HLAVS EXHAUSTED-NUCELL TERMINATION)
END
```

SCHNEIDER,VI  
COPYX

001/68/799

- EFN SOURCE STATEMENT - IFN(S) -

DATE 01.

SUBROUTINE COPY(LLNK,MTOP,MBOT)

C  
C  
C  
C

PUTS ZERO INTO LNKL OF ALL CELLS OF COPIED LIST. IF LLNK = 0,  
AN EMPTY CELL IS RETURNED. IF LNKL (INHALT(LLNK))=0, RETURNS  
LLNK.

COMMON/AVSLX/AVSL, CODE(500)

MTOP = LNKR(AVSL)

KTOP = LLNK

IF(KTOP.EQ.0) GO TO 2051

IF(LNKL(INHALT(KTOP)).GT.0) GO TO 2050

MTOP = KTOP

MBCT=IBTM(MTOP,-1)

RETURN

2050

IKTOP = INHALT(KTOP)

IKTOP1 = INHALT(KTOP+1)

JTOP = NUCELL(DUMMY)

CALL SETIND(ID(IKTOP),0,LNKR(AVSL),JTOP)

CALL STRIND(IKTOP1,JTOP+1)

IF(ID(IKTOP).NE.3.OR.LNKR(IKTOP1).LE.0) GO TO 2052

CALL SETIND(-1,LNKL(IKTOP1)+1,-1,IKTOP1)

2052

MBOT = LNKR(IKTOP)

IF(MBCT.LE.0) GO TO 2054

KTOP = MBOT

2054

GO TO 2050

MBOT = KTOP

RETURN

2051

MBOT = NUCELL(DUMMY)

CALL SETIND(3,0,0,MBOT)

RETURN

END

C

SCHNEIDER,VI 001/68/799  
 OUTOUX - EFN SOURCE STATEMENT - IFN(5) -

DATE 01/03

SUBROUTINE OUTOUT(ITYPE,CONTNT,LOCAT)

DATA IFLSE,ITRUE,ILABEL,IUNDF,IREFRN,IPROCDD/  
 1 5HFALSE,4HTRUE,5HLABEL,6HUNCFND,6HREFRNC,6HPRCCDR/

LOCAT=LOCAT+1

1000 IF(LOCAT.LE.12) GO TO 2000

LOCAT=1

WRITE(6,1050)

1050 FCRMAT(1H,80X)

2000 IF(ITYPE.NE.0) GO TO 2070

GO TO 3000

2070 IF(ITYPE.NE.1) GO TO 2071

IF(CONTNT.EQ.0.0) ICNTNT=IFLSE

IF(CONTNT.EQ.1.0) ICNTNT=ITRUE

GO TO 2090

2071 IF(ITYPE.NE.2) GO TO 2072

ICNTNT=ILABEL

GO TO 2090

2072 IF(ITYPE.NE.4) GO TO 2073

ICNTNT=IUNDF

GO TO 2090

2073 IF(ITYPE.NE.5) GO TO 2074

ICNTNT=IREFRN

GO TO 2090

2074 IF(ITYPE.NE.6) GO TO 2075

ICNTNT=IPROCD

GO TO 2090

2075 IF(ITYPE.NE.7) GO TO 2076

CALL STRDIR(CONTNT,ICNTNT)

2090 GO TO (401,402,403,404,405,406,407,408,409,410,411,412),LOCAT

3000 GO TO (501,502,503,504,505,506,507,508,509,510,511,512),LOCAT

401 WRITE(6,3401) ICNTNT

RETURN

402 WRITE(6,3402) ICNTNT

RETURN

403 WRITE(6,3403) ICNTNT

RETURN

404 WRITE(6,3404) ICNTNT

RETURN

405 WRITE(6,3405) ICNTNT

RETURN

406 WRITE(6,3406) ICNTNT

RETURN

407 WRITE(6,3407) ICNTNT

RETURN

408 WRITE(6,3408) ICNTNT

RETURN

409 WRITE(6,3409) ICNTNT

RETURN

410 WRITE(6,3410) ICNTNT

RETURN

411 WRITE(6,3411) ICNTNT

RETURN

412 WRITE(6,3412) ICNTNT

RETURN

501 WRITE(6,3501) CONTNT



SCHNEIDER, VI  
OUTOLX

001/68/799

- EFN SOURCE STATEMENT - IFN(S) -

DATE 01/0

```

502      GC TO 2076
        WRITE(6,3502)CONTNT
        GC TO 2076
503      WRITE(6,3503)CONTNT
        GC TO 2076
504      WRITE(6,3504)CONTNT
        GC TO 2076
505      WRITE(6,3505)CONTNT
        GC TO 2076
506      WRITE(6,3506)CONTNT
        GC TO 2076
507      WRITE(6,3507)CONTNT
        GC TO 2076
508      WRITE(6,3508)CONTNT
        GC TO 2076
509      WRITE(6,3509)CONTNT
        GC TO 2076
510      WRITE(6,3510)CONTNT
        GC TO 2076
511      WRITE(6,3511)CONTNT
        GC TO 2076
512      WRITE(6,3512)CONTNT
2076     LCCAT=LOCAT+1
        RETURN
3401     FCRMAT(1H+,A6)
3402     FCRMAT(1H+,6X,A6)
3403     FCRMAT(1H+,    12X,A6)
3404     FORMAT(1H+,18X,A6)
3405     FORMAT(1H+,24X,A6)
3406     FORMAT(1H+,30X,A6)
3407     FCRMAT(1H+,36X,A6)
3408     FORMAT(1H+,42X,A6)
3409     FORMAT(1H+,48X,A6)
3410     FORMAT(1H+,54X,A6)
3411     FCRMAT(1H+,60X,A6)
3412     FCRMAT(1H+,66X,A6)
3501     FORMAT(1H+,F12.4)
3502     FORMAT(1H+,6X,F12.4)
3503     FORMAT(1H+,12X,F12.4)
3504     FORMAT(1H+,18X,F12.4)
3505     FORMAT(1H+,24X,F12.4)
3506     FORMAT(1H+,30X,F12.4)
3507     FORMAT(1H+,36X,F12.4)
3508     FORMAT(1H+,42X,F12.4)
3509     FORMAT(1H+,48X,F12.4)
3510     FORMAT(1H+,54X,F12.4)
3511     FORMAT(1H+,60X,F12.4)
3512     FORMAT(1H+,66X,F12.4)
        END

```

0

5

4

0

2

SETIX      DATE 01/03/68      TIME 12-08

PAGE 40

SETI0001

## ENTRY    SETIND

\* THIS FUNCTION SETIND(ID,LNKL,LNKR,CELL) STORES ID IN ID      EULER  
 \* FIELD, LNKL IN LNKL FIELD, LNKR IN LNKR FIELD OF CELL WHOSE      EULER  
 \* ADDRESS IS IN WORD NAMED CELL. IF -1 APPEARS AS      EULER  
 \* ANY PARAMETER BUT CELL, THAT FIELD IS LEFT UNCHANGED.      EULER

|        |      |              |       |
|--------|------|--------------|-------|
| SETIND | STZ  | MM           | EULER |
|        | CLA* | 6,4          | EULER |
|        | STA  | **+9         | EULER |
|        | STA  | **+15        | EULER |
|        | STA  | **+17        | EULER |
|        | STA  | **+18        | EULER |
|        | CLA* | 3,4          | EULER |
|        | TMI  | **+5         | EULER |
|        | ALS  | 30           | EULER |
|        | STO  | MM           | EULER |
|        | CAL  | MASK1        | EULER |
|        | ANS  | **           | EULER |
|        | CLA* | 4,4          | EULER |
|        | TMI  | **+6         | EULER |
|        | ANA  | MASK2        | EULER |
|        | ALS  | 15           | EULER |
|        | ORS  | MM           | EULER |
|        | CAL  | MASK3        | EULER |
|        | ANS  | **           | EULER |
|        | CLA* | 5,4          | EULER |
|        | TMI  | **+2         | EULER |
|        | STA  | **           | EULER |
|        | CLA  | MM           | EULER |
|        | ORS  | **           | EULER |
|        | TRA  | 1,4          | EULER |
| MM     | BSS  | 1            | EULER |
| MASK1  | OCT  | 007777777777 | EULER |
| MASK2  | OCT  | 000000077777 | EULER |
| MASK3  | OCT  | 770000077777 | EULER |
|        | END  |              | EULER |

STRIX DATE 01/03/68 TIME 12-08

PAGE 45

STRI0001

ENTRY STRIND

\* THIS FUNCTION STRIND(DATUM, IADRES) STORES THE VALUE  
\* NAMED BY DATUM IN THE CELL WHOSE ADDRESS IS NAMED  
\* BY IADRES.

|        |      |     |                      |
|--------|------|-----|----------------------|
| STRIND | CLA* | 4,4 | GET ADDRESS OF CELL. |
|        | STA  | **2 |                      |
|        | CLA* | 3,4 | GET DATUM'S VALUE.   |
|        | STO  | **  | STORE IT.            |
|        | TRA  | 1,4 |                      |
|        | END  |     |                      |

STRDX DATE 01/03/68 TIME 12-08

PAGE 50

STRD0001

## ENTRY STRDIR

\* THIS FUNCTION STRDIR(DATUM,CELL) STORES THE VALUE  
\* NAMED BY DATUM IN THE WORD NAMED BY CALL, DATUM  
\* CAN BE EITHER FIXED OR FLOATING POINT AND THE FUNCTION CAN BE  
\* NESTED.

EULER  
EULER  
EULER  
EULER  
EULER

STRDIR CLA\* 3,4 GET DATUM'S VALUE  
STO\* 4,4  
TRA 1,4  
END

EULER  
EULER  
EULER  
EULER

SETDX

DATE 01/03/68

TIME 12-08

PAGE 55

SETD0001

## ENTRY SETDIR

\* THIS FUNCTION SETDIR(ID,LNKL,LNKR,CELL) STORES ID IN ID FIELD, EULER  
 \* LNKL IN LNKL FIELD LNKR IN LNKR FIELD OF WORD NAMED CELL. EULER  
 \* IF -1 APPEARS AS ANY PARAMETER BUT CELL, THAT FIELD IS EULER  
 \* LEFT UNCHANGED. EULER

| SETDIR | STZ  | MM    | INITIALIZE MM           | EULER |
|--------|------|-------|-------------------------|-------|
|        | CLA* | 3,4   | GET THE ID              | EULER |
|        | TMI  | *+5   | TEST FOR NEGATIVE       | EULER |
|        | ALS  | 30    |                         | EULER |
|        | STO  | MM    | STORE SHIFTED ID FIELD  | EULER |
|        | CAL  | MASK1 | MASK OUT THE ID PORTION | EULER |
|        | ANS* | 6,4   | OF CESS                 | EULER |
|        | CLA* | 4,4   | GET THE LNKL            | EULER |
|        | TMI  | *+6   | TEST FOR NEGATIVE       | EULER |
|        | ANA  | MASK2 | BLOCK OUT OTHER FIELDS  | EULER |
|        | ALS  | 15    | SHIFT INTO POSITION     | EULER |
|        | ORS  | MM    | AND SORE IN TEMPORARY   | EULER |
|        | CAL  | MASK3 | MASK OUT LNKL           | EULER |
|        | ANS* | 6,4   | PORTION OF CELL.        | EULER |
|        | CLA* | 5,4   | GET THE LNKR.           | EULER |
|        | TMI  | *+2   | TEST FOR NEGATIVE.      | EULER |
|        | STA* | 6,4   | OVERLAY LNKR OF CELL.   | EULER |
|        | CLA  | MM    | FETCH THE TEMPORARY.    | EULER |
|        | ORS* | 6,4   | STORE THE TEMPORARY.    | EULER |

|       |     |              |       |
|-------|-----|--------------|-------|
|       | TRA | 1,4          | EULER |
| MM    | BSS | 1            | EULER |
| MASK1 | OCT | 007777777777 | EULER |
| MASK2 | OCT | 000000077777 | EULER |
| MASK3 | OCT | 770000077777 | EULER |
|       | END |              | EULER |

CONTX      DATE 01/03/68      TIME 12-08

PAGE 60

CONT0001

|   |   |         |  |       |
|---|---|---------|--|-------|
|   | ENTRY   | CONT    |  | EULER |
|   | ENTRY   | INH-ALT |  | EULER |
| * | DELIVERS THE CONTENTS OF THE WORD WHOSE MACHINE ADDRESS     |         |  | EULER |
| * | IS THE PARAMETER, AND IS STORED AS AN INTEGER. CONT IS USED |         |  | EULER |
| * | IN FLOATING POINT TO PREVENT TYPE CONVERSION AND INHALT IS  |         |  | EULER |
| * | USED FOR INTEGER ARITHMETIC TO FOOL THE SYSTEM.             |         |  | EULER |

|         |      |     |                                   |       |
|---------|------|-----|-----------------------------------|-------|
| CONT    | TRA  | ++1 |                                   | EULER |
| INH-ALT | CLA* | 3,4 | GETS ADDRESS STORED IN PARAMETER. | EULER |
|         | STA  | ++1 |                                   | EULER |
|         | CLA  | **  | GET THE DATA.                     | EULER |
|         | TRA  | 1,4 |                                   | EULER |
|         | END  |     |                                   | EULER |

IDX

DATE 01/03/68

TIME 12-08

PAGE 65

IDX 0001

ENTRY ID

\* THIS PRIMITIVE FUNCTION RETURNS AS AN INTEGER THE ID FIELD  
\* OF THE CELL NAMED AS PARAMETER.

EULER  
EULER  
EULER

ID      CLA\*      3,4      GET THE CELL  
         ARS      30  
         TRA      1,4  
         END

EULER  
EULER  
EULER  
EULER

LNKLX      DATE 01/03/68      TIME 12-08

PAGE 70

LNKL0001

ENTRY LNKL

\* THIS PRIMITIVE FUNCTION RETURNS AS AN INTEGER THE LNKL  
\* FIELD OF THE CELL NAMED AS A PARAMETER.

EULER  
EULER  
EULER

|      |      |              |                         |       |
|------|------|--------------|-------------------------|-------|
| LNKL | CLA* | 3,4          | GET CELL.               | EULER |
|      | ANA  | MASK         | MASK OUT ID AND LNKR    | EULER |
|      | ARS  | 15           |                         | EULER |
|      | TRA  | 1,4          |                         | EULER |
| MASK | OCT  | 007777700000 | 5-7'S FOLLOWED BY 5-0'S | EULER |
|      | END  |              |                         | EULER |



LNKRX

DATE 01/03/68

TIME 12-08

PAGE 75

LNKR0001

ENTRY LNKR

\* THIS PRIMITIVE FUNCTION PRESENTS AS AN INTEGER THE MACHINE  
\* ADDRESS CONTAINED IN THE RIGHT LINK FIELD OF THE CELL NAMED.

EULER  
EULER  
EULER

|      |      |              |                      |       |
|------|------|--------------|----------------------|-------|
| LNKR | CLA* | 3,4          | GET CELL.            | EULER |
|      | ANA  | MASK         | MASK OUT ID AND LNKL | EULER |
|      | TRA  | 1,4          |                      | EULER |
| MASK | OCT  | 000000077777 | 70'S AND 5-7'S.      | EULER |
|      | END  |              |                      | EULER |

MADOVX

DATE 01/03/68

TIME 12-08

PAGE 80

MAD00001

ENTRY MADOV

\* FETCHES THE MACHINE ADDRESS OF THE CELL NAMED AS  
\* A PARAMETER

EULER  
EULER  
EULER

MADOV CLA 3,4  
TRA 1,4  
END

GET THE LOCATION OF THE CELL.

EULER  
EULER  
EULER

JUST DATE 01/03/68 TIME 12-08

PAGE 85

JUST0001

ENTRY JUSTIN  
\* JUSTIN(FL,A,B)=THE BCD NUMBER STORED  
\* IN A AND B IS CONVERTED TO FL.  
\*

|        |      |         |
|--------|------|---------|
| JUSTIN | SXA  | BACK,1  |
|        | PXA  | 0,0     |
|        | LDQ* | 4,4     |
|        | AXT  | 6,1     |
|        | RQL  | 3       |
|        | LGL  | 3       |
|        | TIX  | *-2,1,1 |
|        | LDQ* | 5,4     |
|        | AXT  | 6,1     |
|        | RQL  | 3       |
|        | LGL  | 3       |
|        | TIX  | *-2,1,1 |
|        | SLW* | 3,4     |
| BACK   | AXT  | ** ,1   |
|        | TRA  | 1,4     |
|        | END  |         |

ISHFT DATE 01/03/68 TIME 12-08

PAGE 90

ISHF0001

\* SHIFTS THE ARGUMENT RIGHT ONE CHARACTER AND RETURNS VALUE  
ENTRY ISHIFT

0 ISHIFT CAL\* 3,4  
0 ARS 6  
0 TRA 1,4  
1 END

INTRSX DATE 01/03/68 TIME 12-08

PAGE 93

INTR0001

ENTRY INTRSC

\* COMPARES TWO ALPHANUMERIC WORDS, RETURNING 0 IF THEY MATCH

INTRSC CAL\* 3,4  
ERA\* 4,4  
TRA 1,4  
END

COMPLEMENT OF LOGICAL WORD  
PUT IN AC

EULER  
EULER  
EULER  
EULDE

SCHNEIDER,VI

001/68/799

IBLDR

DATE 01/03/68

|            |       |        |         |        |         |        |         |   |
|------------|-------|--------|---------|--------|---------|--------|---------|---|
| 30. FIOS   | 33715 | .FBD8F | 33621   | EVEN   | 33633   | .DDDFL | 33654   | . |
|            |       | .MQD   | 33657   | .PEX   | 33660   | .FEXP  | 33661   | . |
|            |       | .FIOS. | 33715   | .FSEL. | 34070   | .FILR. | 34074 * | . |
|            |       | .FILL. | 34113   | .FCLS  | 34115 * | .FOPN  | 34121 * | R |
|            |       | .REED  | 34276 * | .BIN   | 34277 * | .FCT   | 34300   | . |
|            |       | .TAP7. | 34361   |        |         |        |         |   |
| 31. FIOH   | 34366 | .FIOH. | 34366   | .FFIL. | 35227   | .FRTN. | 35255   |   |
| 32. FWRD   | 35445 | .FWRD. | 35445   |        |         |        |         |   |
| 33. FRDD   | 35655 | .FRDD. | 35655   |        |         |        |         |   |
| 34. UN05   | 35703 | .UN05. | 35703   |        |         |        |         |   |
| 35. UN06   | 35704 | .UN06. | 35704   | .BUFSZ | 35705   |        |         |   |
| 36. FLOG   | 35710 | ALOG10 | 35710 * | ALOG   | 35711   |        |         |   |
| 37. FXPF   | 36114 | EXP    | 36114   |        |         |        |         |   |
| 38. FXP3   | 36235 | .XP3.  | 36235   |        |         |        |         |   |
| 39. .IOCS  | 36362 | .L(0)  | 36362   | .MONSW | 36402   | .TEOR  | 36451   | . |
|            |       | .CLOS. | 36614   | .ATTC. | 36627   | .SH1   | 37041 * | . |
|            |       | .OP4   | 37152 * | .OP7   | 37203 * | .OP9.2 | 37217 * | . |
|            |       | .READ. | 37272   | .RER1. | 37315   | .WRIT. | 37317   | . |
|            |       | .FEEIT | 37640   | .GTIOX | 37661   | .RW7   | 37777 * | . |
|            |       | .SEL59 | 41065 * | .BSR.  | 41504   | .EOTOF | 41631   | . |
|            |       | .TCHEX | 42173   | .BASIO | 42176 * |        |         |   |
| 40. .IOCSM | 42201 |        |         |        |         |        |         |   |

## I/O BUFFERS

42201 THRU 76705

## UNUSED CORE

76706 THRU 77014

```

.BGIN..BGIN..(  .*  THIS I,  .*  S A DE,  .*  MONSTR,
.*  ATION ,  .*  PROGRA,  .*  M THAT,  .*  HAS ,
.*  BEEN T,  .*  RANSLA,  .*  TED BY,  .*  THE E,
.*  ULER ,  .*  SYNTAC,  .*  TIC TR,  .*  ANSLAT,
.*  CR AND,  .*  IS BE,  .*  ING ,  .*  INTERP,
.*  RETED ,  .*  BY THE,  .*  EULER,  .*  MODIF,
.*  IED PO,  .*  LISH ,  .*  STRING,  .*  INTER,
.*  PRETOR,  .*  .  ).  .OUT. .END. .,  .BGIN..NEW. X
,  .NEW. S  .VRBL.S  .(  .NMBR.202400000000,  .$.  .BGIN..VRBL.
X  .VRBL.X  .IN.  .NMBR.201400000000+  =  .,  .VRBL.S
.VRBL.X  .IN.  )  .IN.  .END. $.  ,  .$.  .(  .*  X=
,  .VRBL.X  .IN.  ).  .OUT. $.  ).  =  .,  .VRBL.X
.VRBL.S  .NMBR.201400000000)  .IN.  =  .,  .(  .*  X=
,  .VRBL.X  .IN.  ).  .OUT. .,  .VRBL.S  .NMBR.202400000000
)  .IN.  .END. .,  .BGIN..NEW. A  .NEW. R  .VRBL.A  .(
.NMBR.201400000000,  .(  .NMBR.202400000000,  .NMBR.202600000000

```

```

).      ,      .NMBR.203400000000).      =      .,      .(      .*      A= ,
.VRBL.A      .IN.      ).      .OUT.      .,      .VRBL.R      .VRBL.A      .NMBR.202400
000000)      =      .,      .(      .*      R= ,      .VRBL.R      .IN.      ).
.OUT.      .,      .(      .*      R.(1)=,      .VRBL.R      .IN.      .NMBR.201400000000
)      .IN.      ).      .OUT.      .,      .(      .*      R.(2)=,      .VRBL.R      .IN.
.NMBR.202400000000)      .IN.      ).      .OUT.      .,      .(      .VRBL.A      .*
R.(1)=      ,      .VRBL.R      .IN.      .NMBR.201400000000)      .IN.      ).
.OUT.      .END.      .,      .BGIN..NEW. N      .NEW. FIBO      .(      .*      N= ,
.VRBL.N      .NMBR.204400000000=      ).      .OUT.      .,      .(      .*      FIBO=
,      .VRBL.FIBO      .$      .FRML.K      .VRBL.K      .IN.      .NMBR.201400000000
.LEQ.      .IF.      .NMBR.201400000000.SWCH..THEN..NOT.      .IF.      .VRBL.FIBO      .LSCL.
.(      .VRBL.K      .IN.      .NMBR.201400000000-      ).      .VRBL.FIBO      .LSCL.
.(      .VRBL.K      .IN.      .NMBR.202400000000-      ).      +      .SWCH..THEN.
.,      $.      =      ).      .OUT.      .,      .(      .*      FIBO(,      .VRBL.N
.IN.      ,      .*      )      =      ,      .VRBL.FIBO      .LSCL..(      .VRBL.N      .IN.
).      ).      .OUT.      .END.      .,      .BGIN..NEW. P      .NEW. A      .NEW. I
.VRBL.P      .$      .FRML.X      .FRML.K      .BGIN..VRBL.K      .IN.      .VRBL.
K      .IN.      .IN.      .NMBR.201400000000+      =      .,      .(      .*      X=
,      .VRBL.X      .IN.      ).      .OUT.      .END.      $.      =      .,      .(      .*
I= ,      .VRBL.I      .NMBR.201400000000=      ).      .OUT.      .,      .(
.*      A= ,      .VRBL.A      .(      .NMBR.203400000000,      .NMBR.204400
000000,      .NMBR.205400000000).      =      ).      .OUT.      .,      .VRBL.P
.LSCL..(      .VRBL.A      .VRBL.I      .IN.      )      .IN.      .VRBL.I
).      .OUT.      .,      .VRBL.P      .LSCL..(      .$      .VRBL.A      .VRBL.I
.IN.      )      .IN.      $.      ,      .VRBL.I      ).      .OUT.      .,      .(      .*
I= ,      .VRBL.I      .IN.      ).      .OUT.      .END.

```

```
.BEGIN..BEGIN..(      .*      ,      .*      ,      .*      ,      .*      ,      .*
,      .*      ,      .*      ,      .*      ,      .*      ,      .*
,      .*      ,      .*      ,      .*      ,      .*      ,      .*
,      .*      ,      .*      ,      .*      ,      .*      ,      .*
,      .*      ,      .*      ,      .*      ,      .*      ,      .*
).      .OUT.
```

(THIS IS A DEMONSTRATION PROGRAM THAT HAS BEEN TRANSLATED BY THE E  
ULER SYNTACTIC TRANSLATOR AND IS BEING INTERPRETED BY THE EULER MODIF  
IED POLISH STRING INTERPRETOR. )

```
.END. .,      .BEGIN..NEW. .NEW. .VRBL..(      .NMBR.,      .$      ,      .$
).      =      .,      .VRBL..VRBL..NMBR.)      .IN. =      .,      .(      .*
,      .VRBL..IN. ).      .OUT.
```

```
(      X=      2.0000)
```

```
.,      .VRBL..NMBR.)      .IN. .BEGIN..VRBL..VRBL..IN. .NMBR.+      =
,      .VRBL..VRBL..IN. )      .IN. .(      .*      ,      .VRBL..IN. ).
.OUT.
```

```
(      X=      3.0000)
```

```
$.      .END. $.      .END. .,      .BEGIN..NEW. .NEW. .VRBL..(      .NMBR.,
.(      .NMBR.,      .NMBR.)      ,      .NMBR.)      =      .,      .(      .*
,      .VRBL..IN. ).      .OUT.
```

```
(      A=      (      1.0000,      (      2.0000,      3.0000
)      ,      4.0000)      )
```

```
,      .VRBL..VRBL..NMBR.)      =      .,      .(      .*      ,      .VRBL..IN.
).      .OUT.
```

```
(      R= REFRNC)
```

```
,      .(      .*      ,      .VRBL..IN. .NMBR.)      .IN. ).      .OUT.
```

```
(R.(1)=      2.0000)
```

```
,      .(      .*      ,      .VRBL..IN. .NMBR.)      .IN. ).      .OUT.
```

```
(R.(2)=      3.0000)
```

```
,      .(      .VRBL..*      =      ,      .VRBL..IN. .NMBR.)      .IN. ).
.OUT.
```

```
(R.(1)=      2.0000)
```

```
.END. .,      .BEGIN..NEW. .NEW. .(      .*      ,      .VRBL..NMBR.=      ).
.OUT.
```

```
(      N=      9.0000)
```

```
,      .(      .*      ,      .VRBL..$      =      ).      .OUT.
```

```
(FIBO= PROCDR)
```

```
,      .(      .*      ,      .VRBL..IN. ,      .*      ,      .VRBL..LSCL..(
.VRBL..IN. ).      .FRML..VRBL..IN. .NMBR..LEQ. .IF. .NOT. .IF. .VRBL.
```



[illegible]



[illegible]

```

.FRML..VRBL..IN. .NMBR..LEQ. .IF. .NOT. .IF. .VRBL..LSCL..( .VRBL.
.IN. .NMBR.- ). .FRML..VRBL..IN. .NMBR..LEQ. .IF. .NOT. .IF.
.VRBL..LSCL..( .VRBL..IN. .NMBR.- ). .FRML..VRBL..IN. .NMBR.
.LEQ. .IF. .NMBR..SWCH..THEN..NOT. .IF. ., $. .VRBL..LSCL..(
.VRBL..IN. .NMBR.- ). .FRML..VRBL..IN. .NMBR..LEQ. .IF. .NMBR.
.SWCH..THEN..NOT. .IF. ., $. + .SWCH..THEN.., $. .VRBL.
.LSCL..( .VRBL..IN. .NMBR.- ). .FRML..VRBL..IN. .NMBR..LEQ.
.IF. .NMBR..SWCH..THEN..NOT. .IF. ., $. + .SWCH..THEN..,
$. + .SWCH..THEN.., $. + .SWCH..THEN.., $. +
.SWCH..THEN.., $. ). .OUT.

```

```

( FIBO( 9.0000, ) = 55.0000)

```

```

.END. ., .BGIN..NEW. .NEW. .NEW. .VRBL..$ = ., .( .*
, .VRBL..NMBR.= ). .OUT.

```

```

( I= 1.0000)

```

```

., .( .* , .VRBL..( .NMBR., .NMBR., .NMBR.).
= ). .OUT.

```

```

( A= ( 4.0000, 9.0000, 16.0000)
)

```

```

., .VRBL..LSCL..( .VRBL..VRBL..IN. ) .IN. , .VRBL.).
.FRML..FRML..BGIN..VRBL..IN. .VRBL..IN. .IN. .NMBR.+ = .,
.( .* , .VRBL..IN. ). .OUT.

```

```

( X= 4.0000)

```

```

.END. $. .OUT.

```

```

( X= 4.0000)

```

```

., .VRBL..LSCL..( $. , .VRBL.). .FRML..FRML..BGIN..VRBL.
.IN. .VRBL..IN. .IN. .NMBR.+ = ., .( .* , .VRBL.
.IN. .VRBL..VRBL..IN. ) .IN. $. ). .OUT.

```

```

( X= 16.0000)

```

```

.END. $. .OUT.

```

```

( X= 16.0000)

```

```

., .( .* , .VRBL..IN. ). .OUT.

```

```

( I= 3.0000)

```

```

.END. Q00000

```

```

#LEGAL COMMAND OR OPERAND. TERMINATE

```